UNIVERSITY OF CANTERBURY

Towards Accurate Earth-Referenced LiDAR Mapping with an Autonomous Mobile Robot

by

Matthew Young

A doctoral confirmation report submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

College of Engineering

Department of Mechanical Engineering

May 2020

Declaration of Authorship

I, Matthew Young, declare that this thesis titled, 'An Investigation of Automating Topographic Surveying Using an Unmanned Ground Vehicle' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Supervisory Team

Senior Supervisor:	Dr Chris Pretty (previously Prof Xiaoqi Chen)
Associate Supervisor:	Prof Richard Green
Industry Supervisors:	Stuart Ralston and Mathias Roehring

Abstract

There is enormous potential in the geospatial industry for mobile robotics to automate terrain mapping. They can reduce operator-induced errors and perform tasks autonomously without supervision. This thesis seeks to quantify the accuracy and speed of an autonomous mapping robot by comparing it to conventional survey methods.

To do this, a proof-of-concept robot was developed from a "Jackal" Unmanned Ground Vehicle (UGV), controlled by the Robot Operating System (ROS). High-accuracy GNSS, IMU and wheel encoder data was fused in an open-source Unscented Kalman Filter (UKF) to localize the robot in UTM coordinates. The ROS navigation stack was used to achieve autonomous navigation.

This robot was used to map two sites, by measuring it's own position as it autonomously navigated between predefined waypoints. This achieved a mean vertical accuracy of 8-17 mm, and mapped the sites in approximately 13-14 minutes. A selection of conventional GNSS RTK, total station and aerial photogrammetry methods by comparison achieved an accuracy in the range 0-70 mm, and took either a few minutes or approximately an hour to complete.

The robot was then upgraded with a LiDAR unit, and a novel method was developed for accurately aligning and registering the scans to produce a point cloud that could be compared to one collected by a scanning total station. This method, termed *Anchor Cloud Mapping* (ACM) was inspired by the methods survey GNSS and total stations use for calibration. The core principles are that the robots trajectory is divided into independent sections, where the clouds in each section are registered using a keyscan, mesh-based, Generalized Iterative Closest Point method. Each cloud is then pivoted around a calibrated stationary robot pose, and aligned to best fit the UKF trajectory. When compared to a Trimble SX10 scanning total station (which is accurate to 2.5 mm), the robot/ACM cloud has a median point-to-point accuracy of 51-59 mm, and collects several times more points which are more evenly distributed throughout the environment. The robot can autonomously survey an area in minutes while the SX10 requires several hours.

Acknowledgements

Thank you to my supervisors: Chris, Richard and Xiaoqi for their feedback and guidance throughout this project. Thank you as well to all of the other postgraduate students at UC who have provided feedback and advice, particularly Josh McCulloch.

Thank you to all the wonderful staff at Trimble New Zealand who have been so helpful and welcoming. Especially to my industry sponsors: Stuart and Mathias, who have not only helped guide the project but provided many wonderful learning opportunities through Trimble. I would also like to specifically thank David Fletcher and Jakub Horejsi for putting up with my incessant questions. I entered this project with no prior knowledge of surveying and they gave an unreasonable amount of their own time to educate me and train me in the proper use of survey equipment.

And finally, thank you to my parents: Kaye and Roger, as well as all my friends who have supported me throughout this journey.

Contents

List of Figures

List of Tables

Abbreviations

ALS	**A**erial **L**aser **S**canner
AP	**A**erial **P**hotogrammetry
CADD	**C**omputer **A**ided **D**esign and **D**rafting
CP	**C**ontrol **P**oint
DEM	**D**igital **E**levation **M**odel
DGPS	**D**ifferential **G**lobal **P**ositioning **S**ystem
DoD	**D**EM (or **D**TM) **o**f **D**ifference
DR	**D**irect **R**eflex
DTM	**D**igital **T**errain **M**odel
EDM	**E**lectronic **D**istance **M**eter
EKF	**E**xtended **K**alman **F**ilter
GCP	**G**round **C**ontrol **P**oint
GICP	**G**eneralized **I**terative **C**losest **P**oint
GIS	**G**eospatial **I**nformation **S**ystems
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
GPS	**G**lobal **P**ositioning **S**ystem
HDOP	**H**orizontal **D**ilution **O**f **P**recision
IMU	**I**nertial **M**easurement **U**nit
ICP	**I**terative **C**losest **P**oint
KF	**K**alman **F**ilter
LoS	**L**oss **o**f **S**ignificance
LiDAR	**Li**ght **D**etection **A**nd **R**anging
NZGD2000	**N**ew **Z**ealand **G**eodetic **D**atum **2000**
PDOP	**V**ertical **D**ilution **O**f **P**recision
ROS	**R**obotic **O**perating **S**ystem
REP	**R**OS **E**nhancement **P**roposal
RMS	**R**oot **M**ean **S**quare
RMSE	**R**oot **M**ean **S**quare **E**rror
RTK	**R**eal **T**ime **K**inematic

SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
TIN	**T**riangulated **I**rregular **N**etwork
TLS	**T**errestrial **L**aser **S**canner
TS	**T**otal **S**tation
UGV	**U**nmanned **G**round **V**ehicle
URDF	**U**nified **R**obot **D**escription **F**ormat
UTM	**U**niversal **T**ransverse **M**ercator
VO	**V**isual **O**dometry
WGS84	**W**orld **G**eodetic **S**ystem 19**84**

Notation

Symbol	Meaning
General Notation	
$a, b, x, ...$	scalar value
$\mathbf{x}, \mathbf{y}, ...$	vector; unless stated otherwise, a column vector
$\hat{x}, \hat{\mathbf{x}}$	predicted or estimated value or vector
$\bar{x}, \bar{\mathbf{x}}$	mean value or vector
$\mathbf{A}, \mathbf{G}, ...$	matrix
$\mathbf{x}^T, \mathbf{A}^T$	transpose of a vector or matrix
$\mathbf{A}^{-1}$	inverse of a matrix
$det(\mathbf{A})$	determinant of matrix $\mathbf{A}$
$\|\|\mathbf{x}\|\|$	norm (length) of a vector
$(...)$	vector with given scalar values
$\{...\}$	set
$\in$	belongs to; $a \in A$ states that a is an element of the set A
$\mathcal{N}(\bar{x}, \sigma^2)$	Gaussian distribution with mean $\bar{x}$ and variance σ^2. Multivariate Gaussian distributions are denoted $\mathcal{N}(\bar{\mathbf{x}}, \mathbf{C})$ with mean vector $\bar{\mathbf{x}}$ and covariance matrix $\mathbf{C}$
$f(), g(), ...$	function
Point Cloud Notation	
$\mathbf{p}_{ij}$	single point represented by a vector containing at three values $[p_{jx}, p_{jy}, p_{jz}]^T$. It is the jth point within cloud i
$\mathbf{n}_{ij}$	normal vector associated with point j within cloud i. It has values $[n_{jx}, n_{jy}, n_{jz}]^T$
A_i	single point cloud with index i
$\mathbf{R}_i$	a 3x3 rotation matrix.
$\mathbf{P}_i$	pose of a cloud as a 4x4 transform matrix. Consists of a rotation matrix $\mathbf{R}_i$ and translation $\mathbf{t}_i$.
$\mathbf{T}$	4x4 transform matrix
$\mathbf{C}_i$	covariance matrix
$\Rightarrow$	represents the direction of point cloud registration; $A \Rightarrow B$ shows that source cloud A is registered to target cloud B

1 Introduction

1.1 Motivation

The advent of electronics and satellite networks has fueled a rapid development in surveying technology. Where once surveying had to be performed with plumb lines, optical theodolites and manual calculations a modern surveyor can use robotic total stations, Unmanned Aerial Vehicles (UAVs) or Global Navigation Satellite Systems (GNSS) to perform the same task at a fraction of the time and cost [1]. So much of the modern survey process has already been automated.

As the capabilities of surveying instruments have grown, so has the demand for greater accuracy. With instruments that are capable of millimeter-level precision [2–5], the method of use now has a greater impact on the measurement error than the instrument itself does. Human operators can introduce a variety of systematic and gross errors in data through mis-calibration, incorrect readings or inconsistencies in survey methodology [6–10]. When propagated over large survey regions these errors can significantly increase the cost of subsequent operations such as road building or earth-moving [11].

Many of the technologies and challenges present in geospatial surveying are also common to mobile robotics [12]. A robot must build a map of the world in order to navigate through it. The more accurate this map is, the more effectively the robot can navigate. Thus it is a natural extension for this map to also be used externally as a terrain map for surveyors. In addition, an autonomous robotic platform equipped with survey-grade equipment has the potential to map an environment faster than a human surveyor with similar equipment. Thereby providing another tool to the surveyor which can be used in situations hazardous to the surveyor, or as an alternative to manual surveying.

These facts show that there is significant potential for mobile robotics to further automate terrain mapping. At the time of writing there has not been extensive research performed which directly compares the performance of mobile ground-based robotics with existing survey methods. This leaves a gap in the current body of knowledge that this research can potentially fill.

1.2 Research Questions

The purpose of this thesis is to investigate if a topographic survey can be automated with an autonomous mobile robot by comparing its performance with several conventional topographic survey methods that are commonly used in industry. Specifically, the question that this research attempts to answer is:

"How does a survey conducted by an autonomous mobile robot compare to a survey conducted by conventional methods?"

This requires establishing what a survey method is, and what instruments are typically used in the survey industry. An autonomous mobile robot must then be developed before it is used to survey a test environment, along with a selection of conventional methods. Geospatial surveying is increasingly moving towards LiDAR scanning to survey an environment. So the second, more specific question this thesis tries to answer is:

"How accurate is a point cloud map created by an autonomous mobile robot compared to a map generated by a conventional scanning survey method?"

Answering this questions requires expanding on the mobile robot prototype to develop an accurate LiDAR-based system for mapping.

1.3 Contributions of the Thesis

The novel contributions of this thesis to the academic community are as follows:

1. A comparison of a mapping robot with conventional survey methods. The comparison in this research provides insights into how speed and autonomy can benefit a mapping robot, while highlighting the potential sources of error that a robot adds.
2. An analysis of loss of significance in cloud registration. This shows how the problem can occur, why it is such a significant issue, and provides practical steps that can be taken to avoid it.
3. A novel method for autonomous LiDAR mapping with a small mobile robot. This method achieves an accuracy that is comparable with survey-grade instruments while collecting more data in a shorter space of time.

1.4 Thesis Overview

Chapter 1: Introduction begins this dissertation by outlining the research purpose, motivation and objectives.

Chapter 2: Background: The Geospatial Survey Process describes several survey instruments and methods that are commonly used in the geospatial industry. These methods are later compared with an autonomous mobile robot in Chapter 6.

Chapter 3: Background: Error in Geospatial Surveying broadly describes the possible sources of error in the survey process, as detailed by existing literature. This chapter is regularly referred to throughout the thesis to explain various sources of error.

Chapter 4: Background: Robotic Mapping describes some common core technologies that mobile robots use to localize themselves in the world and build a map of their immediate environment.

Chapter 5: Design of Proposed Autonomous Mapping System introduces the robotic platform that was constructed in order to test the methods developed in this research. The chapter includes a description of the robot hardware and software, with an emphasis on how autonomy was achieved and how it relates to the surveying context.

Chapter 6: Topographic Surveying with Robot Localization contains the methodology and results of a comparison between the robotic platform and several conventional survey methods.

Chapter 7: Effect of Numerical Imprecision on Mapping describes an under-reported phenomena: Loss of Significance, which is a critical issue when mapping in earth-referenced coordinates. The chapter demonstrates how it affects point cloud registration, and what practical steps can be taken to avoid it.

Chapter 8: Accurate Registration with Sparse Point Clouds tests a mesh-based Generalized Iterative Closest Point (GICP) cloud registration algorithm against its original version which operates on the raw cloud points. The chapter also uses the mesh-based approach to identify which of several point cloud aggregation methods is the most accurate for aligning sequential point clouds.

Chapter 9: Accurate Earth-referenced Mapping applies the research and lessons learned from Chapters 6 - 8 to develop a novel method for creating an earth-referenced map of an environment. The chapter describes this method, and quantifies its accuracy by comparing it to an SX10 total station.

Chapter 10: Conclusion ends the thesis by summarizing the answers to the research question and the key findings of the thesis. This chapter also lists suggestions for future work.

1.5 Publications

This thesis contains work which has been published in the following papers:

- Matthew Young, Xiaoqi Chen, Chris Pretty, Stuart Ralston, and Mathias Roehring. "Development of an autonomous robotic system for terrain mapping" In *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 1-6. IEEE, 2017.
- Matthew Young, Chris Pretty, Sérgio Agostinho, Richard Green, and Xiaoqi Chen. "Loss of Significance and Its Effect on Point Normal Orientation and Cloud Registration." *Remote Sensing* 11, no. 11 (2019): 1329.
- (In peer review) Matthew Young, Chris Pretty, Josh McCulloch and Richard Green. "Sparse Point Cloud Registration and Aggregation with Mesh-based GICP" *Robotics and Autonomous Systems*

2 | Background: The Geospatial Survey Process

This chapter provides background on existing survey instruments and methods commonly used in industry. The chapter begins by clarifying and defining the difference between a survey *instrument* and a full survey *method*. The chapter continues by describing each stage of a survey method and what is commonly used in industry - the instruments and some of the different ways in which the same instrument can be operated.

2.1 Notes on Geospatial Terminology

Datums, Geoids and Projections

There are many different ways of modelling the Earth. A Geodetic datum is a mathematical model of the Earth's shape, also called an ellipsoid [13]. Coordinates relative to the ellipsoid are given in latitude, longitude and height. Latitude and longitude have units of degrees, minutes and seconds. The height is measured in meters above the surface of the ellipsoid.

While an ellipsoid is a model of the Earth's shape, it does not attempt to model the surface of the earth. In addition, because the Earth's tectonic plates move very slowly over time, datums are fixed to a specific point in time. For example, the latest revision of the World Geodetic System was set in 1984, and is known as WGS84. It is maintained by the United States National Geospatial-Intelligence Agency (NGA) and is the official datum of the Global Positioning System (GPS) [14].

In practice, there can be many meters of difference in height between the ellipsoid and sea level [13]. So to provide a reference for zero height, a geoid model is used. This models the Earth's gravitational field and approximates the surface of the ocean as if all tides and currents did not exist [15], i.e. the Mean Sea Level (MSL). The ellipsoid and geoid models are illustrated in Figure 2.1.

Both the ellipsoid and geoid models are spherical, but in many situations it is desirable to work in a Cartesian coordinate system, such as when using a map. This requires distorting the datum, and the mathematical process for doing this is called a projection [13]. As with the ellipsoid and geoid models, there are several different internationally recognized standards for projections. A common projection is the Universal Transverse Mercator (UTM) projection, which divides the Earth into 60 zones and provides a specific projection for each one. UTM coordinates are given as Northing, Easting and Elevation, all measured in meters. This thesis uses the UTM projection and all measurements take place in Zone 59 South.

Elevation vs Height

There are many different models used to represent terrain: Digital Elevation Model (DEM), Digital Terrain Model (DTM), and Digital Surface Model (DSM). These models have originated from different organizations and countries around the world. Lay people may use these terms and others such as "height" and "elevation" interchangeably. This can be misleading as each term has a distinct definition in the survey industry. The term "terrain" refers to height or elevation *in addition to* the location of specific features such as trees, rivers or buildings [16, 17]. The terms "height" and "elevation" refer only to the vertical difference between various mathematical models and the topographic surface being measured.

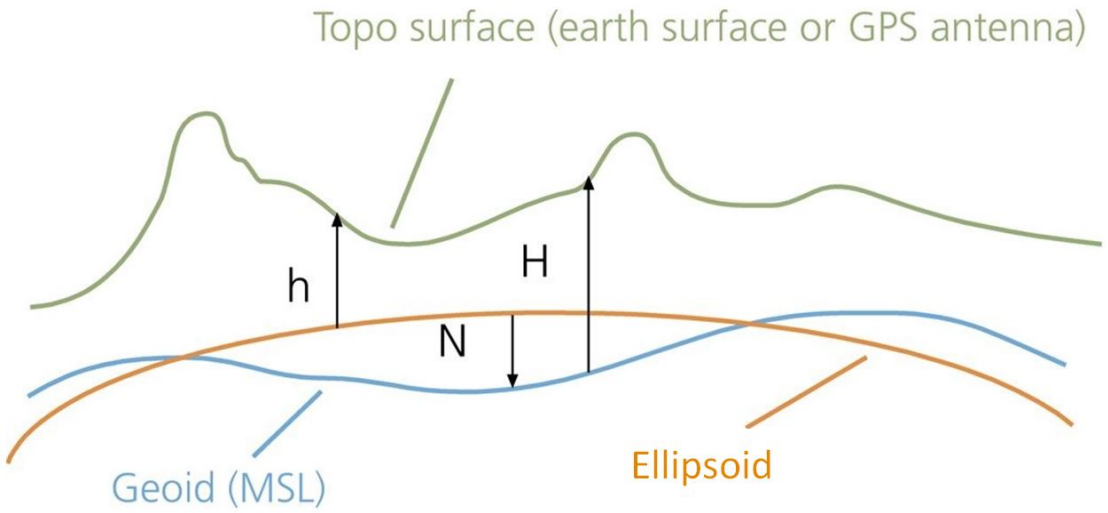


FIGURE 2.1: Illustrative diagram of ellipsoid, orthometric heights and geoid separation.

Source: www.esri.com, 2003 [18]

With specific reference to Figure 2.1, each term is defined as follows:

- **Ellipsoid height (h):** The measured vertical difference between the Earth's surface and the ellipsoid model. This is the sum of the orthometric height and geoid separation (i.e. $h = H + N$) [19, 20].
- **Orthometric height (H):** The vertical difference between the Earth's surface and the geoid [20]. Also commonly referred to as elevation [19].
- **Geoid separation (N):** The vertical distance between the geoid and ellipsoid surfaces. Depending on location this may be positive or negative [19].

So when the *height* of a coordinate is specified, this is typically the ellipsoid height relative to a reference frame such as WGS84. When the *elevation* of a coordinate is specified, this is typically the orthometric height relative to a grid projection such as the UTM grid. This distinction is very important because height and elevation can differ by several meters.

GNSS vs GPS

Global Navigation Satellite Systems (GNSS) is a generic term for any system that uses a network of satellites to provide precise geospatial positioning. Global Positioning System (GPS) is a single GNSS that is operated by the United States Air Force (USAF) [21]. In addition to GPS, there are several other GNSS networks operated by various other nations, such as GLONASS (Russian Federation), IRNSS (India), Galileo (European Union) and BeiDou (China). Modern GNSS systems will often utilize a combination of satellites from different GNSS networks [2].

Accuracy vs Error

Clarification should be made between *accuracy* and *error*. Instruments are often said to be *accurate* to a specific value. This is typically the Root Mean Square Error (RMSE). So in this sense the term *instrument accuracy* can be considered synonymous with "instrument error". However, as stated by Fisher and Tate [8], it can also be argued that if two sets of measurements are collected by different methods, the more accurate of the two can be labelled *accurate* and the other in relative *error*. For this reason, and for the sake of consistency, this thesis will use the term "instrument error" instead of "instrument

accuracy". In addition, all uses of the term "error" are relative to an optimally accurate reference or "ground-truth".

Topographic Survey vs Terrain Map

A *terrain map* is simply a map of a region that shows elevation. This can be created by taking a small number of measurements in key locations, or by scanning the entire region with a photogrammetry or LiDAR device. The term *topographic survey* has a specific meaning in the geospatial industry. The difference between a topographic survey and a terrain map is that a topographic survey also includes the horizontal coordinates of key features in the environment, such as trees, fences, roads, buildings, etc. When a land surveyor surveys an area, they typically conduct a topographic survey. For the purposes of this research, the term *surveying* is used interchangeably with *mapping*, and both refer to creating a basic terrain map, not a full topographic survey.

Earth-referenced map

Most point cloud maps created by Simultaneous Localization and Mapping (SLAM) are given in an arbitrary reference frame, often relative to the starting location of the robot. However, if GNSS information is available, then the map can be provided in coordinates relative a global coordinate system, i.e. *relative* to the *Earth*. An earth-referenced map is then any map expressed in a recognized global coordinate system, such as WSG84 or UTM.

UTM vs `utm`

The robot in this research uses an internal reference frame labelled `utm`, that is centered on the 0,0,0 location of the nearest UTM zone. The reader should be aware that when this term is used in the lowercase Courier font (`utm`), it refers to this internal reference frame. When the term is used in the all-uppercase Computer Modern font (UTM) it refers to the Universal Transverse Mercator (UTM) grid projection.

2.2 Instrument vs Method

Before an analysis of existing survey methods can begin, a distinction must be made between a survey *instrument* and a survey *method*. Many survey instruments have multiple modes of operation and can be used in different ways even within the same operating

mode. The path followed by the surveyor can also be different depending on the environment, instrument and the surveyors discretion. All of these factors can affect the quality of the survey result [7–10, 17]. So a survey instrument in and of itself does not constitute a survey method, and to say otherwise is to oversimplify the survey process.

However, it can be challenging to define exactly what a survey “method" is. Because the natural world is so unstructured and variable, there is no standard operating procedure for a given environment [1]. Nor is there a perfect survey instrument that excels in every situation or a perfect way to operate it. The exact procedure a surveyor follows to conduct a topographic survey will depend on their training, personal experience, and the requirements of the job [1].

This research broadly classifies a survey “method” as the process that begins when a surveyor first steps on the terrain to be surveyed, and ends once all of the raw data has been collected and the surveyor leaves the area. This method can be broken down into four distinct parts:

1. **Survey Instrument:** The instrument that will be used to collect the data, whether it is a manually operated total station or remotely controlled Aerial Laser Scanning system.

2. **Instrument Operating Mode:** The specific mode of operation for the chosen survey instrument, if it has more than one.

3. **Measurement Technique:** The way in which the survey instrument is handled by the surveyor and used to take measurements, e.g a GNSS receiver mounted in a backpack to take “continuous topo” measurements, or on a survey rod in a “stop-and-go” fashion.

4. **Sampling Strategy:** This is the method used to determine the data density and distribution of the survey. For manually operated instruments, this is the physical path followed by the surveyor. Or for other remote-scanning instruments, this is determined by the scan resolution and boundary.

Figure 2.2 illustrates the survey process and breaks it down into each stage. Several common survey instruments are used as examples. The rest of this chapter briefly describes the four components of the survey method in greater detail.

Note that the processing of survey data could be considered part of the survey method as well. However, the goal of this research is to compare human-operated survey methods with an autonomous mobile robot, specifically the differences in how they operate during data collection. Because data processing typically occurs offsite the robot cannot affect it. So the processing tools used are not relevant to this research.

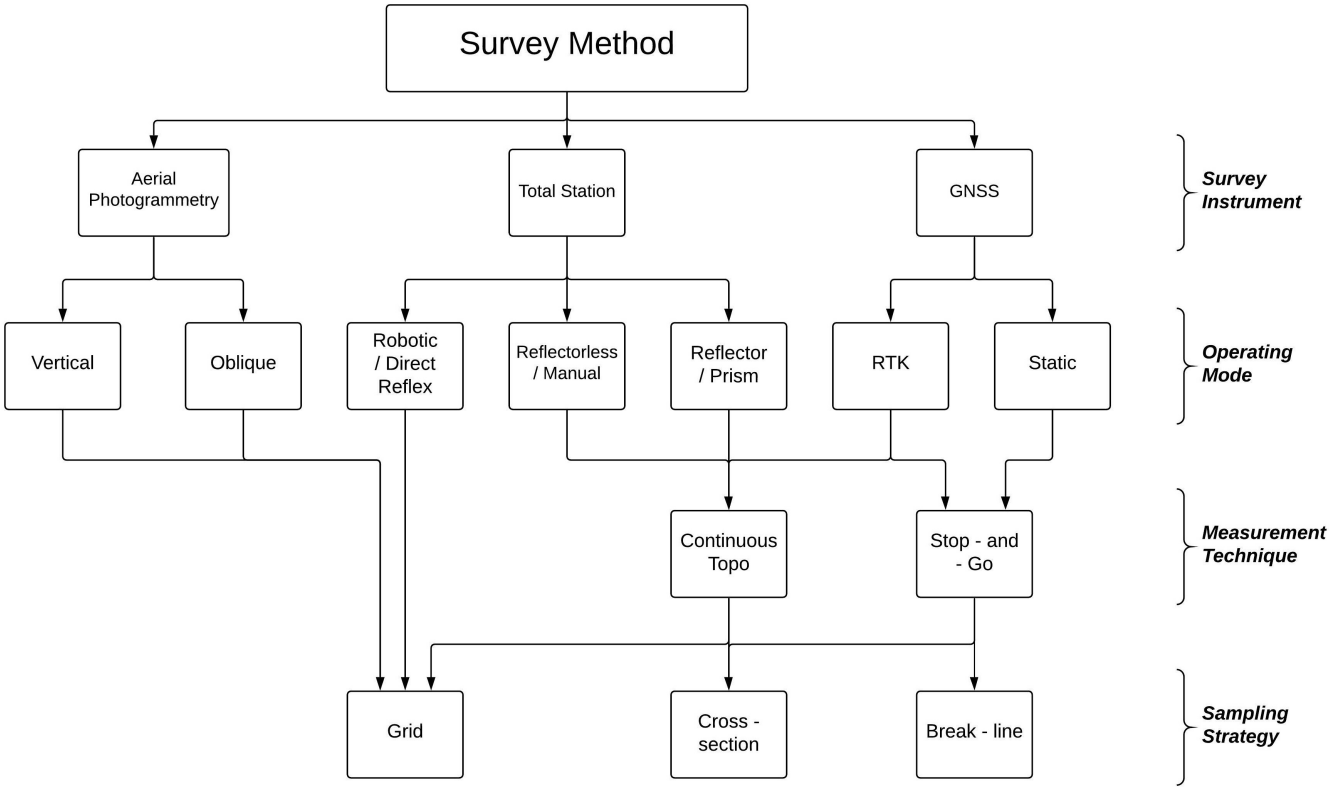


FIGURE 2.2: Example of the four components of a survey method

2.3 Survey Instruments

The first stage of the survey process is choosing a survey instrument. Modern surveyors have a plethora of ground and air based instruments at their disposal. However many of these instruments are relatively new and can be prohibitively expensive for all but professional surveying firms. As a result, most surveys are carried out using either a robotic total station or a GNSS survey system. This section provides brief background on these instruments as well as two common aerial devices.

2.3.1 Total Station

A total station is a device that combines an electronic theodolite with an Electronic Distance Meter (EDM). It can measure the relative angle and distance to any point within its line of sight. If the total station is set up on a known geographical coordinate, this information can then be used to create a topographical map of the surrounding environment. Robotic total stations also include servo-motors, microprocessors and radio or WiFi. This allows them to be controlled autonomously or remotely via a handheld computer. Modern total stations also include built-in memory, so all survey points are stored as they are measured and can be later exported to processing software of the surveyorś choice [22]. All total stations used in this research were robotic total stations.

To set up a total station, the total station itself is mounted on a tripod above a known control point. A tribrach, optical plummet and measuring tape are then used to precisely position the total station and calibrate its height above the control point. Using the control point allows the total station to calculate its exact coordinates. A reflective prism is then set up on another tripod over a different control point, called a "backsight". The total station is then aimed at the prism and it's EDM used to measure the distance between the two. The total station can then use the coordinates of each point and the distance between them to calculate and calibrate its azimuth. The station setup can typically be performed with multiple backsights for greater accuracy.

Modern total stations can be operated in one of three modes [1]: Manual, Prism and Direct Reflex (DR). These modes are also sometimes referred to as reflectorless, reflector and robotic respectively.

Manual Mode

In manual mode the surveyor manually aims the electronic EDM of the total station at an individual point to survey it. This can be done by optically sighting the target with a built in view-finder or by designating a point on a map for the station to automatically target.

Prism Mode

In prism mode the total station does not measure the topography directly, but instead automatically tracks a reflective prism using its EDM and robotic servo-motors. The

prism is mounted on a survey rod which is carried by the surveyor. To survey a point, the surveyor puts the tip of the survey rod on the point, stabilizes the rod so it is as close to vertical as possible, and remotely commands the TS to measure the distance and azimuth to the prism. The length of the rod is then subtracted to calculate the exact coordinates of the tip of the survey rod.

Direct Reflex Mode

In DR mode, the survey defines an area within the total station's field of view for it to automatically survey with it's EDM. The surveyor can specify the exact boundaries of the area and it's grid resolution. The total station will then automatically calculate the required number and position of all the points in the grid, before aiming it's EDM at each one in sequence to conduct the survey.

2.3.2 GNSS Survey System

GNSS survey systems are also widely used in industry. Instead of using angles and distances to measure points relative to a known coordinate, GNSS receivers obtain their latitude, longitude and elevation by communicating with a constellation of satellites [20]. The basic principle is that each satellite broadcasts a signal encoded with the satellite indentification and the time the signal was sent. GNSS receivers on the ground use the time the signal arrived to calculate the distance between receiver and satellite [23]. Connection to four of more satellites allows the receiver to precisely calculate it's global position [22]. Like total stations, GNSS survey systems have built-in memory that stores all measured coordinate data and includes other information such as the quality of the GNSS signal and the number of satellites connected.

In the surveying industry, GNSS systems often use differential positioning (called Differential GPS or DGPS). One receiver is placed on a tripod above a known control point and is referred to as the "base" or "reference" receiver. A second receiver, called the "rover", is mounted on a survey rod, backpack or vehicle and this is the receiver that the surveyor carries around to measure points [22]. Both receivers communicate to establish a vector between them and record their relative positions. This data can be used to remove the clock and atmospheric errors common to both of them and thus significantly improve their positioning accuracy [19].

GNSS systems can have multiple modes of operating depending on the manufacturer, but two of the most universal are Static and Real Time Kinematic (RTK) [1, 22].

Static Mode

Static mode is a very high accuracy operating mode where both the base and rover receiver are left on their respective coordinates for extended periods of time, from a few minutes up to several hours. This gives the position of each satellite (called the satellite geometry) time to change. The change in geometry allows the receiver to more accurately calculate it's coordinates to within a few millimeters. However, the long observation time makes it impractical for topographic surveying.

RTK Mode

Real-Time-Kinematic (RTK) is an advanced form of DGPS. The difference is that RTK measures the satellite carrier wave to further reduce signal error. This gives RTK receivers a greater degree of accuracy over ordinary DGPS systems. RTK can be accurate to a few millimeters [24].

Because of the near-instantaneous observation time of RTK receivers, they are widely used in general purpose surveying and are the instrument of choice for mounting on vehicles, aircraft or other mobile applications.

2.3.3 Aerial Photogrammetry

Photogrammetry works by matching terrain features in overlapping, geo-tagged photographic images to triangulate the location of the feature [25]. When used across a large area this technique can be used to capture the topography of the environment. As the name suggests, Aerial Photogrammetry (AP) applies this technique to photographs taken from an aircraft. Traditionally, AP devices were large and bulky, so they could only be mounted on full-scale piloted aircraft [25]. Recently the technology has become compact and light enough to mount on a variety of fixed-wing and multi-rotor Unmanned Aerial Vehicle (UAVs).

Many current aerial photogrammetry systems still use non-differential GNSS systems because of their smaller scale and weight. Such receivers are typically only accurate to a few meters and give coordinates in a global datum such as WGS84. To achieve

very high accuracy, large fiducial markers (shown in Figure 2.3) are placed around the survey area prior to take off. These are referred to as Ground Control Points (GCPs) and they are used to calibrate the height of the UAV [25] and act as a reference point if the coordinates need to be converted to a local datum (such as NZGD2000). However, newer UAV systems have integrated RTK receivers that can reduce or eliminate the need for GCPs [26].

FIGURE 2.3: Example of aerial photogrammetry GCPs

Although aerial survey systems are becoming increasingly more common, they are subject to aviation laws which can restrict their use [26]. As an example, for this research, inquires were made regarding the use of a fixed-wing UAV for surveying the test grounds. However the fixed-wing UAV could not be used because it's wide turning arc would require it to pass over a nearby motorway, which is prohibited by local aviation laws. As a result, a multi-rotor UAV had to be used instead.

AP can be considered to have two different operating modes, depending on the angle at which the photographs are taken: oblique and vertical photogrammetry, as illustrated in Figure 2.4. While the mathematics used in each case are very similar, their applications are typically different [27].

It should also be noted that other forms of photogrammetry exist. Terrestrial and space photogrammetry are also used in surveying, but as neither of these options were available for this research, they have been omitted.

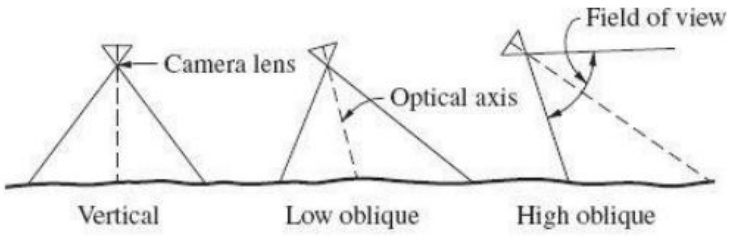


FIGURE 2.4: Aerial Photogrammetry angles

Source: Wolf 2000 [27]

Oblique Photogrammetry

Oblique photogrammetry is conducted with photos intentionally taken at a non-perpendicular angle to the ground. These images can be further divided into high oblique photos, where the horizon is visible, and low oblique where it is not. These photos provide a perspective more familiar to a human viewer, and are more often used for illustrative or modelling purposes [28, 29]. The disadvantage of oblique photogrammetry is that many more photos taken from different directions are required to fully map a large objects such as buildings.

Vertical Photogrammetry

Vertical photogrammetry is conducted with photographs taken perpendicular or nearly perpendicular to the ground. This provides a better view of enclosed spaces like city streets. Vertical photogrammetry is more commonly used in mapping applications for generating DEMs and orthophotos (geometrically corrected images) [27]. Figure 2.3 was taken using vertical photogrammetry.

2.4 Measurement Techniques

Some survey instruments can be used to take measurements in different ways, even within the same operating mode. For example, a robotic total station in prism mode can be used to take measurements while the prism is held stationary (stop-and-go), or at set intervals while it is moving (continuous topo) [19, 24]. Each measurement technique affects the data in very different ways so it is important to define them and their differences. By comparison, scanning instruments measure terrain directly from the device, typically using photogrammetry or LiDAR.

These devices operate in only one way, regardless of the surveyor, so they do not strictly have a measurement technique as defined by this research.

Most modern survey systems include functionality to support both measurement methods [24], and both methods are widely used in industry. This project will use and compare both for the sake of completeness and because the continuous topo technique closely resembles how the autonomous UGV conducts a survey. The rest of this section will describe each technique in greater detail.

2.4.1 Stop-and-Go

When using a stop-and-go measurement technique a surveyor will physically stop at each survey point, place the instrument on the point, and stabilize it before taking a measurement. In this scenario the surveyor will typically be using GNSS receiver or a prism mounted on a survey rod.

2.4.2 Continuous Topo

When using a continuous topo measurement technique the surveyor does not stop at individual points when surveying, instead simply walking or driving over the area to be surveyed with the survey instrument set to take measurements at regular intervals. This is also sometimes referred to as "kinematic" mode [19]. Continuous topo is generally more flexible than the stop-and-go technique, although it can be performed with the exact same setup. This measurement technique is typically used with GNSS receivers mounted in back packs or on vehicles.

2.5 Sampling Strategy

The term "sampling strategy" refers to how the surveyor chooses which specific points to survey. Rather than simply choosing points at random, a surveyor will generally follow a specific path or pattern across the survey area. With ground-based survey instruments this may be a literal path along a ridge, field or river bed. With remote-surveying instruments such as TLS or ALS the sampling strategy refers to the boundary and resolution of the region to be scanned. Choosing an appropriate strategy is an important

part of the survey process because it dictates the data density and distribution of the survey, which in turn affects the survey accuracy [9, 30–32], as discussed in Chapter 3.

Ultimately the sampling strategy is up to the surveyors own experience and discretion, and there is no perfect way to survey any specific environment. This section will now discuss three of the more common sampling strategies: grid, break-line and cross-section.

2.5.1 Grid

A grid-based survey, as the name suggests, surveys every point in a regular grid defined by the surveyor. All the surveyor needs to specify is the boundary and the resolution of the grid. A different surveyor can re-create the same topographic map without requiring the exact coordinates of each point the first surveyor measured. This survey path inherently provides an even distribution of data across the whole survey area which means that, unlike the other survey paths, the coverage of a topographic map is not dependent on the surveyor's prior training or experience. The disadvantage of this method is that using a gird-based survey for a large area can be unfeasible or prohibitively time-consuming. In addition, regardless of the gird resolution, critical points such as local maxima and minima will rarely occur at grid points, thus reducing overall accuracy [6].

Remote-surveying instruments that use laser devices or aerial photogrammetry typically survey points in a grid pattern by default. However, algorithms used in post-processing the data may eliminate some points, resulting in an irregular point distribution, as shown in Figure 2.5.

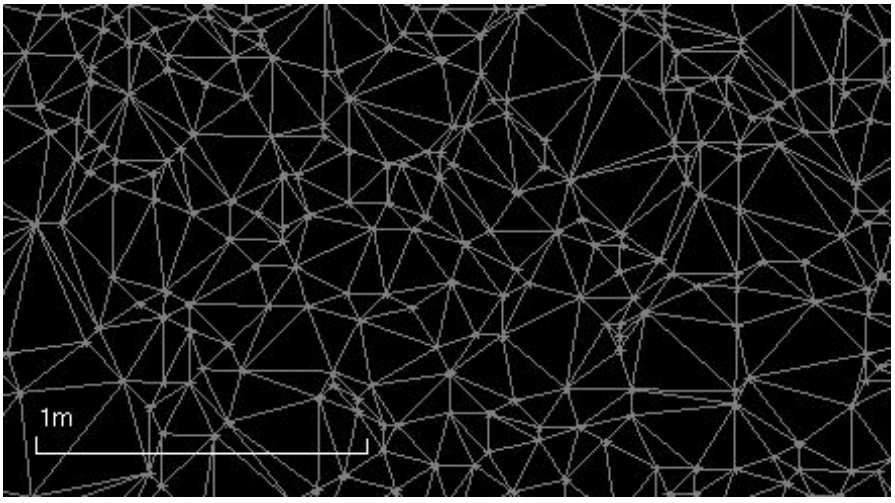


FIGURE 2.5: Close-up view of a surface created from aerial photogrammetry data

2.5.2 Break line

A break line survey exploits the natural contours of the environment to produce a topographic map with a minimum of survey measurements. A break-line is defined as a continuous feature which represents a sharp change in slope such as a ridge, riverbed, wall or road-edge. When conducting a break-line survey, a surveyor will take elevation measurements along multiple break-lines to properly define the topography. Break-lines may often run perpendicular to the contours of the terrain to properly capture the slope of a hill or trench. Each individual break-line is chosen at the surveyor's discretion, and can be as long or as short as the surveyor deems necessary. Figure 2.6 shows a topographic model produced from a break-line survey.

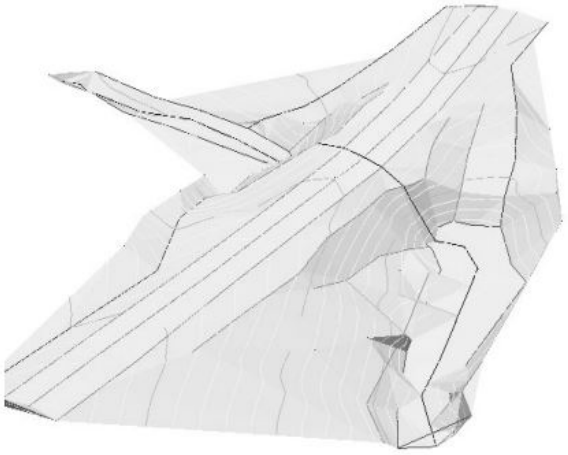


FIGURE 2.6: Example DEM modeled using a break-line survey

Source: University of York [33]

A break line survey can be a very fast and efficient compared to other methods and can be used to quickly survey very large areas. However, as can be seen in Figure 2.6, the resulting model can appear "blocky" due to the uneven distribution and low density of data. For this reason, the accuracy of a break-line survey is highly dependent on the skill and experience of the surveyor. This also means that two surveyors could survey the same area, choose different break-lines, and produce two different maps. Regardless, this sampling strategy is commonly used in industry.

2.5.3 Cross-section

A cross-section survey path is similar to a break-line survey with the key difference being that the survey follows only one major break-line (often called a baseline). The baseline typically follows a long geographic feature such as a ridge or river bed, and may be broken up by intermediate points which indicate a change in direction of the break-line, as represented by the points "PC" and "PT" in Figure 2.7 [1]. Measurements are taken along lines perpendicular to the baseline to capture the slope or elevation of the feature [1].

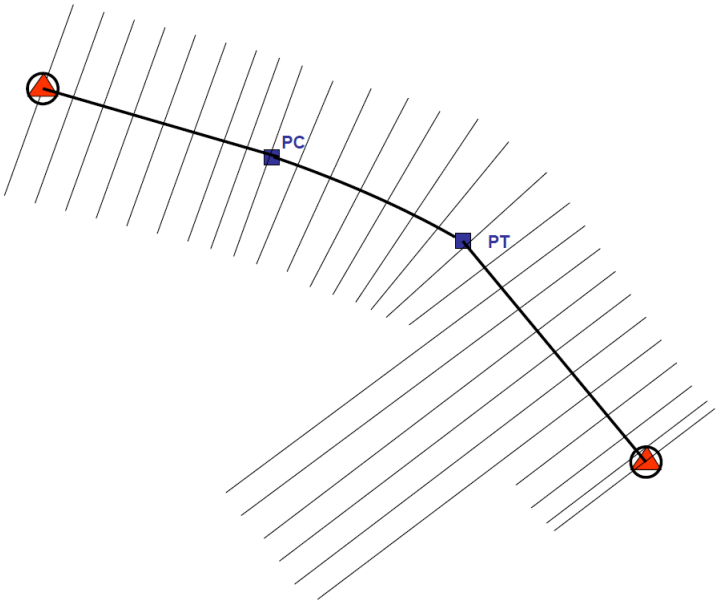


FIGURE 2.7: Example of a cross-section baseline

Source: USACE, 1994 [1]

Like a break-line survey, the cross-section survey results in a non-uniform distribution of data which in turn creates an non-unifrom distribution of error. Heritage *et al.* [9] notes that the error in a DTM produced using this strategy is lower close to cross section points and higher in areas with little measured data. In practice, the cross-section survey path is rarely used in industry [1].

2.6 Processing Survey Data

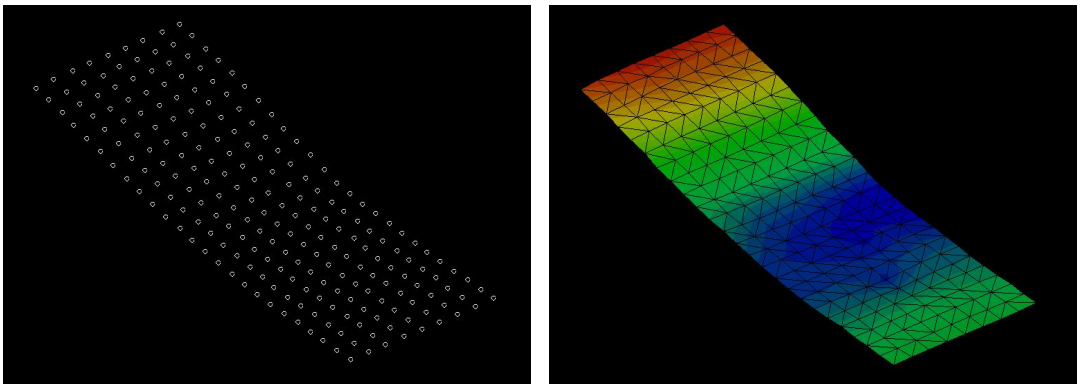
Once collected, most raw survey data is processed in Computer Aided Design and Drafting (CADD) or Geospatial Information System (GIS) software. This software is collectively used to correct, edit, splice and display the data in meaningful ways that can be more easily interpreted or used to aid in subsequent operations. Generally the data will be filtered first, to remove noise or gross errors from the raw data. The raw data is generally not meaningful by itself so it must then be processing into something that spatially represents the survey.

The specific result depends on the end user, but in most cases at least one of the final products will be some form of topographic map as they are one of the easiest ways to represent spatial data. A Digital Elevation Model (DEM) is one of the most basic topographic maps which, according to the definition previously outlined in Section 2.1, simply shows the elevation of the terrain at each point in the map.

Creating any sort of topographic surface requires interpolating between the measured data points. There are a large variety of different interpolation methods [34]. Some of the most common are:

- Kriging (with many sub-variations)
- Nearest-Neighbour
- Triangular Irregular Network (TIN)
- Inverse Distance Weighting (IDW)
- Minimum Curvature

These methods use a variety of algorithms to estimate the information between measured data points, and each will produce a slightly different result [9]. The differences and advantages of each method are well documented [9, 16, 34–37] and a complete analysis is beyond the scope of this research. Regardless of the specific interpolation method used, the end result is a surface model similar to the one shown in Figure 2.8.



(a) Measured points (b) Interpolated DEM

FIGURE 2.8: Example of a DEM surface created using TIN interpolation.

2.7 Summary of the Survey Process

As explained in this chapter, a survey *instrument* is simply a device. The way in which it is used is called a survey *method*. A number of different survey methods exist, which are based around GNSS receivers, total stations or aerial platforms. Because these survey methods are used later in this thesis, the material in this chapter and the next is helpful for explaining their respective sources of error. The way in which these instruments are set up also inspires some of the novel methods developed later in this thesis.

3 | Background: Error in Geospatial Surveying

A significant proportion of the motivation in robotics and automation comes from the desire to reduce error in a process. Therefore, it is important to understand the possible sources of error in that process. This chapter provides a background on some of the different sources of error in a topographic survey. The chapter defines the error metrics commonly used in industry and describes how each error can be broadly categorized. The rest of the chapter briefly describes several common survey errors by source.

The chapter does *not* provide an in-depth analysis of each error, nor does it provide an exhaustive list of all possible sources of error. Doing so would be impractical as the level of error depends on the type, brand and use of the survey instrument. The purpose of this chapter is simply to explain some possible sources of error in surveying, why they matter and provide the motivation for this chapter as well as some of the key decisions made within it. For more detailed and specific analysis, the reader should read the cited sources where they are provided.

Many possible sources of error are discussed in this chapter, and the reader could be forgiven for thinking that if they all had an effect, every survey measurement should be wildly inaccurate. The reality is that error in the survey process is well documented, and there are many ways of mitigating or eliminating most sources of error if care and due diligence is exercised. Surveying is not a question of "how accurately can an area be surveyed?" but "what is required for this job, and is it worth the time and effort to achieve that degree of accuracy?"

3.1 Definition of Accuracy, Error and Uncertainty

There are many ways of defining error: by magnitude, by type (e.g. gross, systematic, random), or by source. For the purposes of this research, error will be defined by source. In the surveying context, error can be roughly classified by source into one of four main categories:

1. **Instrument Error:** This is the error introduced specifically by the instrument itself during the measurement process. Typically a result of sensor noise, imperfect mathematical models or aging in the instrument.

2. **Operation Error:** Also called "human" or "personal" error [6], this is introduced directly by the survey team as a result of improper use of survey equipment or flawed survey methodology. This also includes mis-calibration of the instrument by the surveyor.

3. **Environmental Error:** Also called "natural error", this is introduced by any aspect of the terrain or natural environment that directly or indirectly obstructs the use of the survey instrument. This includes factors such as wind, rain, temperature, atmospheric pressure and refraction, gravity and so on.

4. **Post-processing Error:** Error introduced by processing or interpolation methods after the raw data has been collected. Typically introduced when raw data is used to create a surface map.

Some errors do naturally overlap between categories, and these errors will be discussed where appropriate. Also note that some textbooks and papers consider issues such as mis-calibration to be a instrument error rather than operation error [7]. However, as this error stems from the *operator* and not the *instrument*, this research classifies it as an operation error.

There is some ambiguity between the terms instrument *accuracy* and instrument *error*. For clarification on the use of these terms in this chapter, refer to Section 2.1: Notes on Terminology.

3.1.1 Metrics of Error

Several metrics exist for quantifying different types of error in a geospatial survey. A common metric is the Mean Error (ME), which is the arithmetic average of all the residual errors in a DEM and is therefore used to quantify the bias in the results. The ME in surveying is defined by Equation 3.1.

$$ME = \frac{\sum (Z_{DEM} - Z_{Ref})}{n} \tag{3.1}$$

Where:

Z_{Ref}	=	Elevation of a high accuracy reference or "ground truth" point
Z_{DEM}	=	Elevation of a DEM created from measured points
n	=	The number of sample points

The ME alone is not a sufficient measure of accuracy. A set of elevation measurements may be wildly different from the expected value, but if the sum of positive error values is similar in magnitude to the sum of negative values, then the mean error will still be small. By contrast, the Root-Mean-Square Error (RMSE) is the square root of the mean quadratic error, and therefore still captures the error if the bias (ME) is small. For this reason it is a better measure of accuracy. The RMSE is defined by equation 3.2.

$$RMSE = \sqrt{\frac{\sum (Z_{DEM} - Z_{Ref})^2}{n}} \tag{3.2}$$

Many researchers also use the Standard Deviation (SD) as a measure of precision, which measures the average quadratic deviation from the mean. Note that if the ME is very small, the SD is approximately equal to the RMSE. Equation 3.3 defines the SD.

$$SD = \sqrt{\frac{\sum ((Z_{DEM} - Z_{Ref}) - ME)^2}{n-1}} \tag{3.3}$$

Typically, just the RMSE will be used in manufacturers data sheets to specify the accuracy of a survey instrument, and is typically split into a "horizontal" and "vertical" component. Academic research may also use a variety of other metrics to describe the performance of survey equipment [10, 38, 39].

3.2 Instrument Error

Instrument error is the error introduced by the instrument itself during the measurement process. No instrument is perfect and there are practical limits on how accurately an instrument may be able to determine its position due to sensor noise, approximations in mathematical models or finite limits on resolution. This information is well defined by the manufacturer and is provided as the rated accuracy on the instrument's data-sheet, so it is known from the beginning of the survey. Assuming minimal error from other sources, this is the best or expected accuracy with the instrument. As an example, Table 3.1 lists several modern survey instruments and their rated accuracy.

TABLE 3.1: Example of modern instrument accuracies

		Accuracy (RMSE)	
Instrument	Operating Mode	Horizontal	Vertical
VX Total Station[1] [3]	Prism mode	2 mm + 2 ppm	2 mm + 2 ppm
	DR & manual mode	2 mm + 2 ppm	2 mm + 2 ppm
R10 GNSS System[1] [2]	Differential	250 mm + 1 ppm	500 mm + 1 ppm
	Static	3 mm + 0.5 ppm	5 mm + 0.5 ppm
	RTK	8 mm + 1 ppm	15 mm + 1 ppm
UX5 HP UAV[2] [5]	Photogrammetry	10 - 20 mm	10 - 50 mm

All instruments shown are manufactured by Trimble.
[1] Parts Per Million (ppm) is added inaccuracy for every million units (in this case millimeters) between the the survey point and total station/GNSS base station
[2] Photogrammetry accuracy is dependant on image resolution, which is configurable. This tables assumes the smallest resolution of 10 mm.

For all survey instruments used in this research, the baseline between GNSS RTK rover and receiver, or total station and measured point was small enough that the ppm component of each error can be considered negligible. Specific examples of instrument error include:

- **Zero Error:** As prisms age their reflector constant can change, which affects their measured distance from a total station EDM or other laser [22]. Typically on the order of a few millimeters, this error can be mitigated by calibrating and re-measuring the prism constant.
- **Cyclic Error:** Errors can occur in measuring the phase difference between the transmitted and received laser pulse in an EDM. This creates errors in the distance

measurement and has a variety of external and internal causes. The eventual error in the distance measurement varies and can be on the order of a few millimeters. This error can be eliminated via calibration [22].

- **Scale Error:** Total station EDMs depend on a crystal oscillator to output laser pulses at specific frequencies. Age, damage or variations in temperature can affect the frequency of the crystal and the ability of the EDM to accurately calculate distances. This error is proportional to the distance from EDM to prism (hence "scale" error) and can be on the order of 1 - 5 ppm [22]. This error can be eliminated via calibration.

- **Clock Error:** Errors in the clock of the GNSS receiver or satellite can result in large errors in distance calculations [40], 3000 km for every 0.01 s of error [22]. Errors in the satellite are carefully monitored, modelled and corrected by the satellite operator [40] while errors in the receiver can be mitigated by differential positioning or by connecting to at least four satellites.

- **Wear and Warpage:** As survey rods age they may wear down at the base or become warped, distorting their effective height [41]. This produces a systematic error and can be corrected by periodically checking and re-measuring the rod.

3.3 Operational Error

Operational error encompasses all the sources of error that directly result from the improper use of survey equipment by the surveyor. This may be through mishandling the instrument, recording an incorrect measurement, mis-calibration or poor choice of survey methodology. Most operational errors can be avoided with due care and by repeating or double-checking measurements.

However, even the best professional surveyors cannot perfectly replicate the same procedure every time. Variations in how equipment is operated will occur, not only with the same surveyor but also between different survey teams. As an example, Bangen (2013) provides a case study in which five survey teams surveyed the same six sites with similar instruments and techniques. The results of the study show that different crews often produce different results for the same site, with the mean difference ranging from 0.05 - 0.67 m depending on the site. At individual points however, different crews could

produce results with a difference of several meters, as high as 10.05 m [10]. This serves as an example of how operation error can affect the quality of a topographic survey.

Operational error can be further divided into three sub-sections: calibration, measurement (also called "gross" error) and survey strategy error.

3.3.1 Calibration Error

Calibration error is a systematic type of operator error introduced by improper equipment calibration and set up. As with other types of error, the scale is different depending on the instrument and what component was mis-calibrated. Examples of calibration error include:

- **Incorrect Control Point:** TSs use the control point (CP) they are set up on as a reference for their global position, so any horizontal or vertical error in the CP will be added to every subsequent TS measurement. This error is systematic and can be removed in post-processing or avoided altogether with multiple, thorough GNSS measurements of the CP.
- **Incorrect backsight:** Backsights are critical for calibrating the azimuth of TSs. If the backsight CP has not been surveyed properly, it will add an error to every TS measurement that is proportional to the azimuth error and the distance from the TS. This can be mitigated by setting up a backsight a significant distance from the TS or by using multiple backsights.
- **Incorrect TS leveling:** If the TS is not perfectly level any inclination in the vertical axis will cause a varying inclination in the horizontal axes, depending on the azimuth of the TS [6]. Many modern TSs have tilt sensors on both the vertical and horizontal axis, and will either automatically correct for any angular errors, or warn the surveyor of them [22].
- **Incorrect Rod Height:** Incorrectly entering the rod height will create an offset in every subsequent measurement. This occurs through a variety of individual measurement errors (which are discussed separately in Section 3.3.2). While the scale of the error is highly random, it is systematic and can often be removed in post-processing.

- **Mis-calibrated bubble-level:** If the bubble-level is not accurate then the TS, GNSS base station or survey rod will not be perfectly vertical, introducing variable error in all measurements with that instrument. The scale of the error depends on the device the level is mounted on, and the degree of error in the level's calibration. Bubble-levels typically need to be re-calibrated on a regular basis as hard knocks or temperature extremes can distort the calibration [24].

3.3.2 Measurement Error

The most obvious and most common form of error during a topographic survey will be through simple measurement error. In the surveying context, this will typically be a gross or random error introduced when measuring a single coordinate. Or it may be an error made at the beginning of the survey which creates an ongoing systematic error. These errors only apply to manual survey methods, as automated systems like AP or TLS don't require human input during operation.

Many sources of measurement error are intuitive and not necessarily specific to survey equipment, examples include:

- **Reading Error:** Measuring the wrong value on a scale, either by not reading the correct value or by not transcribing it correctly into a notebook or computer. For example writing '1.50' instead of 1.05'. This kind of error can only be corrected with diligence and double-checking.
- **Parallax Error:** Incorrect reading of a scale due to the angle of perspective. This can be easily avoided by ensuring that the scale is read at eye-level.
- **Leveling Error:** If a survey rod is not held level it will introduce a horizontal and vertical error that is proportional to the rod's angle from true vertical. This localized error applies only to that measurement and is generally on the order of only a few millimeters. This error is a distinct problem for the continuous-topo measurement technique, as it is virtually impossible to hold the rod perfectly vertical while moving. For stop-and-go measurement, this error can be easily avoided with due care and a calibrated bubble-level. Some modern GNSS receivers also incorporate IMUs to measure and compensate for any tilt.

3.3.3 Survey Strategy Error

Survey strategy error is not error in the same sense as misreading a value or miscalibrating an instrument. The "error" is not having enough measured data to properly capture the topography of the survey area [8, 17]. This happens if the data is not dense enough or not properly distributed around key features, such as ridges or river banks. If all the local maxima and minima have not been adequately surveyed, large errors can be introduced between the DEM and reality when the data is interpolated [9, 42]. Obviously there are practical limits on the time and ability of a survey team and their equipment, so it is impossible to perfectly capture the topography of an environment. This starts to be called an "error" only when there are large gaps in the data or the quality does not meet the requirements of the job.

In general, errors in sampling strategy are avoided through experience and training. Many survey manuals and survey companies have their own checklists and Standard Operating Procedures (SOPs) for field surveying [1], which improve quality. Some examples of survey strategy errors are:

- **Low Sampling Effort:** Insufficient time spent surveying a particular region can result in a low density of measured points. This is a particular issue for earth-moving jobs if a hill or depression in the ground has not been properly surveyed [11]. This can be corrected by spending more time on site and is typically a cost/accuracy trade off.
- **Low Image Resolution:** Resolution in aerial photos determines the size of the smallest feature that can be identified and matched in overlapping images. This in turn determines density and accuracy of the resultant data [43]. In large-scale projects (particularly involving photos taken from orbit) the accuracy of the survey result may be so low that it cannot be used for anything other than visual inspection [17]. In these cases the data may not be suitable for the job and another source of data is required (e.g. from a ground-based survey team).
- **Poor Choice of Break-Line:** The break-line and cross-section sampling strategies rely on a well-chosen line that follows a continuous, dominating feature of the environment, such as a river or mountain ridge. A poor choice of break-line or cross-section offsets may fail to properly capture this feature.

3.4 Environmental Error

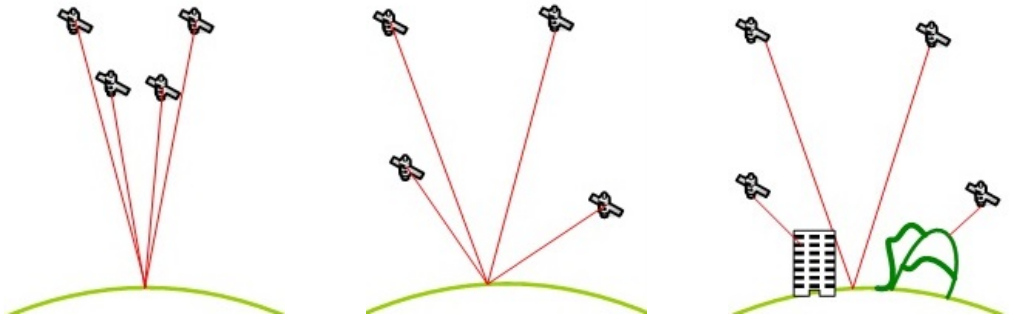
Environmental error is any error that is directly introduced by the environment the surveyor is working within. The environment in general is one of the least controllable sources of error, as surveyors may often be required to work to deadlines that require them to conduct surveys regardless of the weather. Environmental errors can be further divided into two sub-catagories: environmental variation and environmental obstruction.

3.4.1 Environmental Variation

Environmental variation errors are caused by a random and constantly changing aspects of the environment, such as weather. They can directly affect internal components or obstruct signals and line of sight.

GNSS survey systems are especially prone to variations in the environment, since they depend on accurate satellite signals and communication with base stations. There are a few sources of error that are particularly notorious specifically for GNSS devices:

- **Ionospheric Effects:** The Ionosphere is a layer of the Earth's atmosphere that is filled with ionized particles, which can disrupt GPS signals [23, 44]. The effect is dispersive, so the level of error depends on the frequency of the signal and can be up to several meters. Most of the error can be negated with a variety of mathematical models [40], which are periodically updated by the satellites themselves [45].
- **Troposphere Effects:** The Troposphere is the lowest layer of atmosphere that contains all of the Earth's weather systems. It has varying levels of pressure, temperature and water vapor, which can disrupt signal quality and increase error by up to half a meter [46]. This error can also be corrected with accurate mathematical models and predictions of typical delays [23, 40].
- **Dilution of Precision:** Satellite geometry (the arrangement of satellites in orbit above the GNSS receiver) can have a significant effect on GNSS accuracy. There is always some degree of uncertainty in the signal range from a satellite. A broad arrangement of satellites (Figure 3.1(b)) provides a better position estimate than when they are clustered together (Figure 3.1(a)) or obscured (Figure 3.1(c)) [23, 47].



(a) Bad geometry, high DOP (b) Good geometry, low DOP (c) Good geometry, poor visibility, high DOP

FIGURE 3.1: Possible satellite geometries

OneSky [48]

The exact level of uncertainty is called the Dilution Of Precision (DOP), and is used as a measure of the quality of the available satellite configuration. DOP can be divided into more specific components: Horizontal (HDOP), Vertical (VDOP) and Positional (PDOP). Table 3.2 provides a general guide of the value ranges and the corresponding quality.

TABLE 3.2: DOP quality chart

Quality	DOP
Ideal	<1
Excellent	1-2
Good	2-5
Moderate	5-10
Fair	10-20
Poor	>20

Satellite constellations changes over time as the satellites move. For this reason many sites have optimal times of the day to conduct a survey, when the DOP is lowest [7].

Ionosphere, Troposphere and DOP effects are far more complex than what is briefly described here, readers wishing to learn more should consult a dedicated GNSS textbook such as *GPS Satellite Surveying* by Leick, Rapoport and Tatarnikov or *GPS: Theory, Algorithms and Applications* by Xu and Xu. Other examples of error due to environmental variation include:

- **Wind:** Strong wind gusts can introduce vibrations in tripods or make it difficult to keep a survey rod steady. The level of error introduced varies depending on the weight of the tripod, the force of the wind and the strength of the surveyor.
- **Tripod Settlement:** If set up on unstable or soft ground, the tripod may sink into it over time, introducing a systematic error to every measurement [7, 41]. This can be easily avoided with careful placement or use of leveling plates to distribute the weight of the tripod [22]. This type of error could also be considered a form of operational error.
- **Hot Spots:** Haze and reflective surfaces such as snow or water can create "hot-spots" where light back-scatters into the camera lens of an aerial photogrammetry unit. This degrades the contrast and detail of any photo taken, thereby reducing accuracy [25, 49].
- **Temperature:** Survey instruments are designed to operate inside a certain temperature band. However, instruments left in direct sunlight can heat up unevenly, causing parts to expand at different rates and creating small errors [1].

3.4.2 Environmental Obstruction

With many instruments, environmental error can occur when some terrain feature physically obstructs accurate use of the instrument. In many cases this is due to some permanent feature of the environment such as a building, tree or other vegetation.

- **Multipath Error:** This occurs when GNSS satellite signals reflect off large surfaces, often nearby buildings (as illustrated in Figure 3.2) or the ground, before reaching the receiver [23, 40]. This artificially increases the transmission time of the signal and can result in erroneous calculations of the receivers location [6, 44, 50]. In extreme cases the multipath error can be severe enough that the receiver is forced to ignore the signal altogether [20].

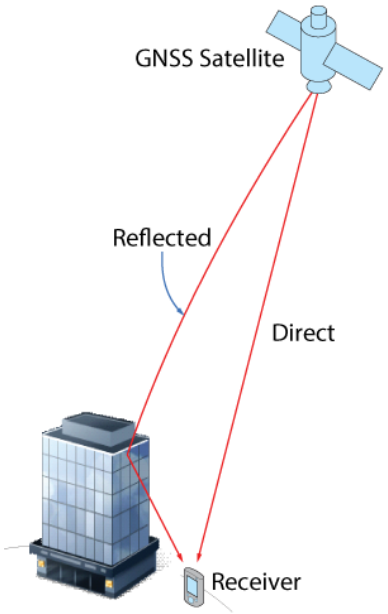


FIGURE 3.2: Illustration of multipath error

Source: www.novatel.com, 2016 [46]

- **Line of Sight Blocking Terrain:** Because the GNSS signal is very weak, large features such as trees can easily block GNSS signals from some satellites altogether, reducing accuracy [20, 44, 51]. Intervening terrain is a common issue for scanning instruments as well. TSs in DR mode, TLS and ALS all require a direct line of sight to the target in order to survey it. This can be difficult in especially dense or mountainous terrain. In many cases even light vegetation can obstruct view of the terrain, making it artificially appear to be higher [8]. Sometimes vegetation can be so dense that it totally prohibits use of scanning instruments, such as in forests and jungles. Research indicates that even such factors as the tree species and whether or not it has leaves can affect the accuracy [51, 52].

- **Fog and Haze:** Physical weather phenomena such as fog, haze and clouds disrupt aerial photogrammetry by distorting the view of the ground or obscuring parts of it altogether [25]. These events can cause minimal error or they can make a survey impossible to complete.

3.5 Post-Processing Error

Once the raw survey data has been collected it has to be processed into meaningful information, often a DEM or other surface map. Error can be introduced during processing by noise, filtering, rounding or other mathematical errors.

Some examples of post-processing error are:

- **Interpolation Error:** Section 2.6 has already explained how different methods can be used to create a DEM by interpolating measured data. No interpolation method can perfectly estimate the topography between points, since interpolation algorithms inherently assume that the terrain will be continuous when in many cases it will not be. As a result there will always be some level of error between the interpolated surface and the true topography [8]. The scale of error is variable, depending on the interpolation method, data density and terrain complexity [35]. As a result different interpolation methods may be used on the same source data and produce DEMs with different accuracies.

- **Photogrammetry Processing Error:** As explained in Section **??**, photogrammetry works by matching features in overlapping images. Modern AP software automates this process but errors can occur due to noise, blurring, lens distortion or error in the position of the GCPs. These errors degrade the quality of the image which can cause the software to make false matches or mis-calculate the distance between features [43, 53], thereby introducing error.

- **Datum Transformation Error:** Many different geodetic datums exist for modelling the earth's surface, such as the World Geodetic System 1984 (WGS84) and the North American Datum 1983 (NAD83). Precise mathematical equations can be used to transform coordinates between datums. But these datums are not always fixed relative to one another and can drift over time. This means that the transformation equations can become outdated or obsolete, introducing errors in the process. [54].

3.6 Summary of Error in the Survey Process

This chapter has briefly described just a few of the possible sources of error that can be introduced during the survey process. In each case this chapter barely scratches the surface, many sources of error in surveying are complex and their solutions even more so. Several studies have been dedicated to one specific source of error and readers wishing to learn more should consult the relevant literature.

Some generalizations can be made based on this brief analysis. While all sources of error are well studied, it is often only the predictable sources of error that are well minimized. Instrument errors are relatively low as they happen within the (relatively) controlled environment of the instrument, so they can be reliably modelled and mitigated. Environmental errors tend to be well-studied but cannot always be perfectly mitigated because of their inherently random nature. This is why some forms of environmental error, such as vegetation obstructing instrument line of sight, still present challenges for survey teams. This in turn, is dependant on operational error and the surveyor. Operator error is the least predictable form of error in the survey process, and can only be mitigated through training, experience and diligence. This gives motivation to the project and shows that there is potential for mobile robotics to improve the quality of the topographic survey. This can be done by making more reliable measurements and by automating the sampling strategy.

4 | Background: Robotic Mapping

4.1 Map Representation

The way a map is represented in a Simaltaneous Localization and Mapping (SLAM) process is often determined by the application. If the primary objective of SLAM is to localize the observer for navigation purposes, then the map is often represented in a discretized format such as an elevation map or voxel grid. This is often the case with autonomous robots or self-driving cars, as the map is not a desired output but simply a means for the robot or car to navigate the world.

However in situations where the map itself is the desired product, a point cloud is often used. Because the desired product is the map itself, discretizing the scans into an elevation map or voxel grid would result in a loss of fidelity.

No representation is perfect, and all have their own advantages and disadvantages. This Section will briefly cover some of the more common 3D map representations, such as those shown in Figure 4.1.

(a) Elevation Map (b) Voxel Map (c) Point Cloud

FIGURE 4.1: Different 3D representations of a tree

Source: Hornung *et al*, 2013 [55]

4.1.1 Elevation Map

An elevation map is not a true 3D map, but a "2.5D" map. It is a 2D map with limited vertical information. Each horizontal coordinate has one or more vertical values associated with it, but does not represent the full vertical space at that coordinate. These maps are based on the assumption that all space below the vertical value is occupied and all space above is not. This makes elevation maps inherently more computationally efficient than full 3D representations, while still capturing vital 3D information.

Some modifications of an elevation map store the elevation as probabilistic value rather than an absolute one. Instead of storing a single value, an elevation mean and variance is stored instead. Multiple vertical measurements for a single horizontal coordinate are typically combined using an algorithm such as a Kalman Filter [56].

After determining the vertical height for a particular horizontal coordinate (whether as a single or probabilistic value), most elevation map implementations only store that height. This means that it is impossible for an elevation map to accurately represent multi-level geometry such as overhanging cliffs, roofs, bridges or trees [57] (as shown in Figure 4.1(a)). To overcome this limitation, some researchers have used modified elevation maps which contain multiple vertical values for each horizontal coordinate. These are referred to as "extended" or "multi-level" elevation maps. These maps usually incorporate multiple vertical values but do not explicitly represent all vertical space [58, 59]. Also note that this definition of a *robotics* elevation map is not to be confused with a *surveying* elevation map, which is simply a point cloud or 3D surface that shows the height of the land.

4.1.2 Voxel and Octree Maps

A voxel map is the 3D extension of a 2D grid or occupancy map. Each "voxel" is a cube in 3D space, which has one of three states: *free*, *occupied* or *unknown*. These states can be discrete (each voxel is occupied or not) or they can be probabilistic (each voxel has a probability of being occupied).

Unlike elevation maps, each vertical and horizontal space is explicitly represented (compare Figure 4.1(b) to Figure 4.1(a)). This makes voxel grids well-suited for 3D navigation, as obstacle-free space is explicitly differentiated from un-mapped space.

However, there are several disadvantages to voxel maps, notably their computation and memory requirements. The size and resolution of voxel grids typically needs to be defined *before* an area is mapped. If the required resolution is too small, the voxel map can become prohibitively slow to compute. But if the resolution is too low (i.e. the voxels are too big), the map may overly simplify the environment.

An octree is essentially just a voxel map with a tree-like data structure and variable voxel sizes. Each voxel is represented by a "branch" node and can be further subdivided into eight smaller voxels or "leaf" nodes [60]. Each node can be subdivided over and over to achieve a finer spatial resolution. Like ordinary voxels, octree voxels can be modelled as discrete [61] or probabilistic objects [62–64].

Octrees share several advantages with voxel maps, such as the fact that they implicitly represent free space and differentiate it from unmapped space. And like voxel maps, their explicit 3D representation makes them ideal for 3D navigation.

However, octrees are more efficient than an ordinary voxel map because a branch node can be "pruned" if all of it's leaf nodes have the same state. In practice this means that large empty or occupied regions of the map can be pruned and reduced to a small number of large voxels. Note that multiple resolutions of voxels can exist within the same map, as demonstrated in Figure 4.2. This makes octree-based maps far more computationally efficient than ordinary voxel maps [55].

4.1.3 Point Cloud Map

A point cloud is not a "map" in the same sense as an elevation or octree map. A point cloud is simply an unstructured collection of points in 3D space. For robots using LiDAR or other range finders, processing point clouds directly can save time and computation that would have otherwise been spent converting the data into a discretized map. As examples, several researchers have developed SLAM algorithms that operate directly on point clouds [65–69].

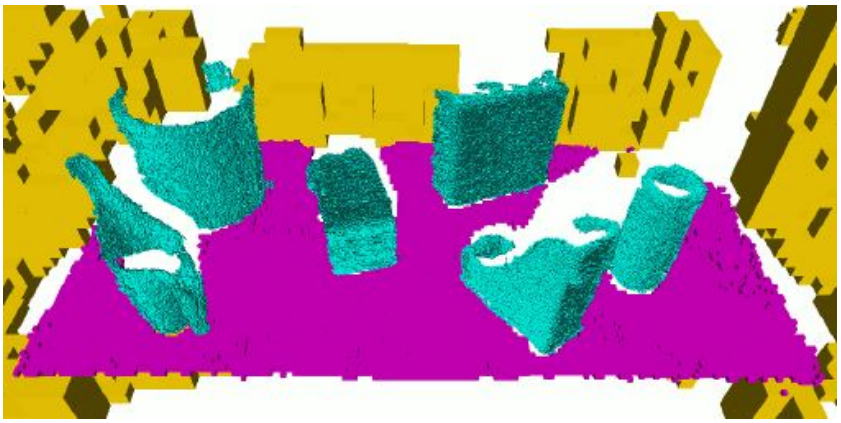


FIGURE 4.2: Example of multiple resolution voxels within one octree map

Source: Hornung *et al*, 2013 [55]

One advantage of point clouds is that they are loss-less maps. No data has been lost by combining measurements into a discrete form (such as a voxel) which can never have a resolution as fine as the raw data. This makes point clouds preferable for accurate mapping. The downside to this approach is that the complexity and size of the cloud has no limit, so it's storage requirements and computational complexity will only increase over time.

4.2 Point Cloud Registration

Point cloud registration is the process of aligning two point clouds such that their common features match. For this reason the process is also sometimes referred to as *scan-matching*. There is a section of academic research dedicated to solving this problem, and the variety of solutions is large. Some algorithms explicitly look for common features such as edges, planes or corners in both clouds and compute the transform that minimizes the distance between them. Other algorithms, such as the family of algorithms known as Iterative Closest Point (ICP) operate by iteratively pairing two points, one in each cloud, and moving the clouds in a way that minimizes the distance between pairs.

Most cloud registration algorithms must be tuned to some extent. At a minimum this usually means tuning the stopping criteria. But it can also include parameters in the algorithm itself, or in point filters used to refine the points to register. Generally tuning is done by hand, although some researchers such as [70] have used parameter optimization methods to find the best set of parameters for their application.

This section will describe how point normals are computed (because this is an important first step in some ICP variants), how three common variants of ICP work, as well as some methods used to combine point clouds after registration.

4.2.1 Point Normal Vector Calculation

To maintain consistency with existing literature, this thesis employs the same notation adopted by Klasing et al. [71] and Jordan et al. [72], who denoted a cloud of N points as $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N\}, \boldsymbol{p}_i \in \mathbb{R}^3$, each point as a triplet $\boldsymbol{p}_i = [p_{ix}, p_{iy}, p_{iz}]^T$ with an associated normal vector $\boldsymbol{n}_i = [n_{ix}, n_{iy}, n_{iz}]^T$. Any given point $\boldsymbol{p}_i$ for which we want to calculate the normal of has a neighborhood of k points $Q_i = \{\boldsymbol{q}_{i1}, \boldsymbol{q}_{i2}, ..., \boldsymbol{q}_{ik}\}, \boldsymbol{q}_{ij} \in P, \boldsymbol{q}_{ij} \neq \boldsymbol{p}_i$.

The normal vector $\boldsymbol{n}_i$ for any given point $\boldsymbol{p}_i$ in a cloud is determined by its surrounding neighborhood of points Q_i, which together with $\boldsymbol{p}_i$ typically approximate some shape or laser-scanned surface. In some applications, the cloud is given as a square or triangular mesh, in which case the point normals are referred to as vertex normals, and can be calculated directly from the mesh using neighboring triangles or vertex angles. [73]. However, in many cases involving real-world data such as that produced by a LiDAR unit, this mesh is not available and the point normals must be calculated from the structure of the cloud itself.

Most normal calculation methods operate by fitting a plane [74], surface or other geometric model [75] to $\boldsymbol{p}_i$ and its neighbors Q_i. Errors between points and the model are minimized using processes such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA), and then it is from the model that the normal vector is derived. A good explanation and comparison of such methods can be found in the works of Klasing et al. [71] and Jordan et al. [72]. Some methods may be modified by applying weights [74, 76] or bounds [77] to improve the accuracy of the solution.

However, while these studies often discuss the accuracy and computational speed, they do not discuss the stability of these methods, or how they can be affected by numerical precision and the ensuing consequences. So at the time of writing, there appears to be no research which directly addresses numerical precision and its effect on point cloud operations such as point normal calculation and cloud registration.

4.2.1.1 Calculation of Normals with Eigenvalues

The most widely used method to calculate the normal for a point is to use a Least-Squares formulation to fit a plane to the point $\boldsymbol{p}_i$ and its k-nearest neighbours. The normal is the vector of values $[n_{ix}, n_{iy}, n_{iz}]^T$ that best satisfy the equation for a plane, given by Equation 4.1.

$$an_{ix} + bn_{iy} + cn_{iz} + d = 0 \tag{4.1}$$

This process requires calculating a 3-dimensional co-variance matrix for the cluster of points, the standard form of which given by Equation 4.2, with variance on the diagonal and co-variance on the off-diagonal.

$$\mathbf{C} = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(x,y) & cov(y,y) & cov(y,z) \\ cov(x,z) & cov(y,z) & cov(z,z) \end{bmatrix} \tag{4.2}$$

Where $cov(x, y)$ describes how the x component of the point coordinates varies with their y component, $cov(x, z)$ how x varies with z and so on. This matrix encodes the variance of the points in 3D space, where each eigenvector lies along a principal component of the cluster and the corresponding eigenvalue represents the magnitude of the variance in that direction. If the points approximate a plane, their coordinates will vary along the length and width of the plane significantly more than in the direction of its depth or "thickness". Therefore, the smallest eigenvalue of $\mathbf{C}$ corresponds to the eigenvector that lies orthogonal to this plane, in the direction of least variance. The normalized eigenvector is the point normal. This process is critical for different variations of the Iterative Closest Point algorithm, which are discussed in the next section.

This method of computing the normal vector was first proposed by Hoppe et al. [74] in 1992 and this method, or a close variation of it, has been used in numerous studies since [71, 75, 77–79].

4.2.2 Iterative Closest Point

One of the most widely used methods for matching point clouds is the Iterative Closest Point (ICP) algorithm. First introduced by Besl and McKay [80], the goal of ICP is to match or "register" two point clouds (often referred to as the "source" and "target" cloud). In the simplest form of this problem, two clouds A and B have m points in the sets $A = \{\boldsymbol{p}_{ia}, ..., \boldsymbol{p}_{ma}\}$ and $B = \{\boldsymbol{p}_{ib}, ..., \boldsymbol{p}_{mb}\}$ respectively. Where $\boldsymbol{p}_{ia}$ and $\boldsymbol{p}_{ib}$ are corresponding points. The standard "Point-to-Point" ICP algorithm then finds a transform T that aligns the clouds by minimizing an error function E shown in equation 4.3.

$$E(\boldsymbol{T})_{ICP} = \frac{1}{m}\sum_{i=1}^{m} \|\boldsymbol{T}\boldsymbol{p}_{ia} - \boldsymbol{p}_{ib}\|^2 \tag{4.3}$$

ICP algorithm starts with an initial guess for each match and then iteratively improves the estimate of $\boldsymbol{T}$ by re-matching pairs and re-computing the error function, eventually converging to an optimal solution.

In practice this problem is often more complicated. The clouds may be non-identical, may sample the same object at different points ($\boldsymbol{p}_{ia} \neq \boldsymbol{p}_{ib}$), may only partially overlap, may contain many sources of error and so on.

To deal with this, an alternative version of the ICP algorithm models the surface between neighbouring points in cloud B and then minimizes the distance between that surface and the points cloud A. This is referred to as "Point-to-Plane" ICP (PICP) [81], and modifies the ICP error function as shown:

$$E(\boldsymbol{T})_{PICP} = \frac{1}{m}\sum_{i=1}^{m} \|(\boldsymbol{T}\boldsymbol{p}_{ia} - \boldsymbol{p}_{ib})\boldsymbol{n}_{iab}\|^2 \tag{4.4}$$

Where $\boldsymbol{n}_{iab}$ is orthogonal to the modelled surface in B and runs between the surface and point $\boldsymbol{p}_{ia}$. This algorithm intelligently minimizes the error along the direction of the surface normals, while allowing the clouds to slide in directions orthogonal to the normals. The latest version of the ICP algorithm extends this further by using co-variance matrices of point neighbourhoods to model and align the cloud surfaces directly:

$$E(\boldsymbol{T})_{GICP} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{d}_i^T (\boldsymbol{C}_i^B + \boldsymbol{T}\boldsymbol{C}_i^A \boldsymbol{T}^T)^{-1} \boldsymbol{d}_i \tag{4.5}$$

Where $\boldsymbol{C}_i^B$ and $\boldsymbol{C}_i^A$ are the co-variance matrices of corresponding points $\boldsymbol{p}_{ia}$ and $\boldsymbol{p}_{ib}$ and $\boldsymbol{d}_i = \boldsymbol{p}_{ib} - \boldsymbol{T}\boldsymbol{p}_{ia}$ is a vector of their point-to-point distances. Note that here $\boldsymbol{d}_i^T$ and $\boldsymbol{T}^T$ are transposes. This is called "Plane-to-Plane" or "Generalized" ICP (GICP) [82]. By using co-variance matrices of point neighbourhoods directly, GICP does not calculate or require normal vectors. Many other variations of the standard ICP algorithm exist, however standard ICP and PICP and are the most widely used within the research community. Note that a point neighbourhood is any group of neighbouring points, where the bounds of the neighbourhood may be defined by a radius from a central point, or by the k-nearest points.

The full ICP algorithm can be broken down into approximately five stages: Point selection, pair matching, weighting, rejection and calculation of the error function [83]. Each stage can be separately augmented to improve accuracy or speed. For example, using k-d trees when matching points can significantly improve efficiency [84, 85]. Or, during calculation of the error metric, point normals can be used to calculate a "point-to-plane" [81] or "plane-to-plane" [82] distance to be minimized. For a comprehensive analysis of ICP variations and their effect, readers should refer to Rusinkiewicz and Levoy [83].

4.2.3 Registration of Sparse or Non-uniform Point Clouds

Achieving highly accurate point cloud registration can be a difficult task, even more so when the point clouds are irregular. Many popular 3D LiDAR designs have a small number of LiDAR emitter and receiver pairs, with limited vertical resolution. In addition, it is still common for some researchers to use a 2D LiDAR device in an actuated housing to create 3D point clouds [86–90]. In both instances the result is a sparse or non-uniform density cloud, where points are grouped in vertical or horizontal rings.

This makes successful cloud registration difficult for a several of reasons. First, the point density may not be high enough in some parts of the cloud to accurately calculate the point normals or co-variances, which are necessary for several registration methods. Second, the ringed nature of the clouds means that two consecutive clouds may have scanned different parts of the environment with little overlap, even if the clouds are

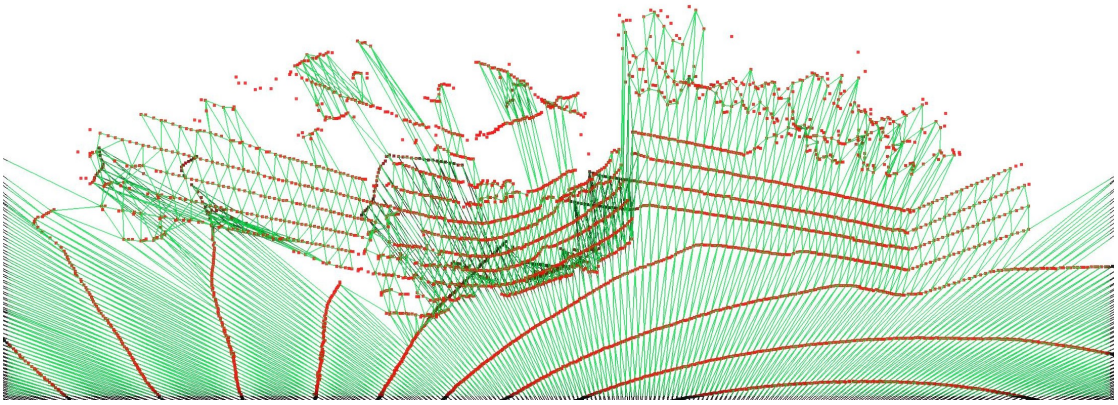
relatively close. Third, incorrect point correspondence can lead to clouds being matched along scan-lines, as illustrated in [86]. For these reasons, even state-of-the-art registration algorithms like Generalized ICP (GICP) [82] may routinely fail to accurately register sparse LiDAR scans.

The fundamental problem is that a sparse or non-uniform cloud does not adequately represent the underlying topography. Many solutions attempt to bypass this problem by fitting planes or other geometric features to the available points. For example, LiDAR Odometry and Mapping (LOAM) uses k-d trees to detect planar surfaces and edges in each newly acquired cloud [88]. Other methods use: a Hough-based voting scheme [91], a region-growing algorithm [90, 92, 93] or Principal Component Analysis [89] to identify and register planes rather than the raw cloud. Beyond these examples, there is a significant body of research on plane and feature extraction. These are merely some of the examples that apply this research to SLAM or mapping.

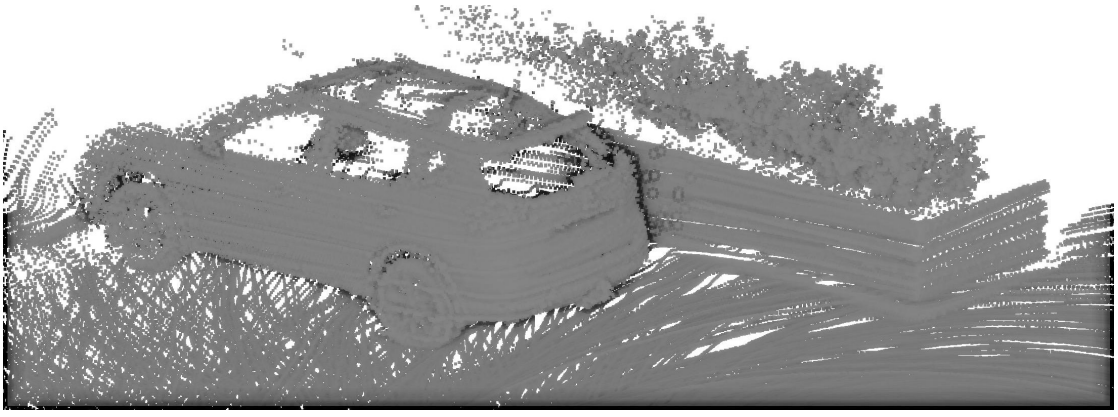
Because of their dependence on planes, these methods are limited to highly structured environments where a sufficient number of planes can be extracted. Instead, the works of Holz and Benke [86, 87] simplify this process by organizing the raw cloud and connecting adjacent points to form a mesh. The mesh is then used in the GICP algorithm to approximate the co-variances $\boldsymbol{C}_i^B$ and $\boldsymbol{C}_i^A$ rather than using the raw points. Compared to alternative solutions, this method is more accurate [70] and has a very low computational cost [86]. Holz and Benke do not give their modified GICP method a unique name, so this thesis will refer to it as mesh-GICP (MGICP) hereafter to distinguish it from unmodified GICP. Figure 4.3 illustrates an example of a single organized point cloud, and the mesh created from it.

4.2.4 Methods for Point Cloud Aggregation

Multi-view registration (also called the "registration strategy") refers to the process by which a set of overlapping point clouds are aligned with each other. Historically, this term has applied to the registration of dense, uniform range images, where each image is a different angle of the same object (e.g. a statue or teapot). The term also applies to the registration of point clouds in mobile robotics, mapping and SLAM, but is under-researched in this area. Most research uses one of the following methods or a close variant:



(a) Single organized point cloud and corresponding mesh



(b) Registered result

FIGURE 4.3: Example point cloud of a car created by registering sequential Velodyne HDL-32 scans with mesh-based GICP

1. **Pairwise:** Each newly acquired cloud A_i is registered to the previous cloud $A_i \Rightarrow A_{i-1}$.

2. **Metascan:** (also called 'incremental') Every new cloud A_i is first registered and then concatenated to a larger meta-cloud $A_i \Rightarrow M$, which consists of all previously registered clouds $M = \{A_j \mid j \in 1 \ldots i-1\},\ i > 1$.

3. **Keyscan:** The set of all clouds is divided into sub-sets, where one cloud in each sub-set is fixed (the 'key' scan A_k) and all other clouds in the sub-set are registered to it $A_i \Rightarrow A_k,\ \{A_i,\ A_k \mid i, k \in\ n \ldots m\},\ i\ \neq\ k$.

These are sometimes referred to as "registration methods" in literature. Thereby confusing them with error minimization algorithms such as GICP. For clarification, the remainder of this thesis will refer to them as *aggregation* methods.

Point cloud registration, and then aggregation forms the "front-end" of many SLAM systems. The way the "back-end" then optimizes the position of the clouds is more varied and beyond the scope of this thesis.

Of the three methods, pairwise is the most commonly used aggregation method as it integrates well with graph-based optimization algorithms. Whereas the typical result of the metascan and keyscan methods is a large, aggregated point cloud. In such cases, if the original poses of the constituent clouds are not retained, then any alignment errors cannot be undone at a later time.

Wulf *et al.* [94] compare pairwise and metascan aggregation methods using point-to-point ICP with and without the popular 6D Lu and Milos (LUM) [95] graph optimization. Their results show that the metascan methods consistently outperform their pairwise equivalents in terms of position and orientation error, but take considerably longer to compute [94]. For this research, registration was performed on 3D point clouds, but the position and rotation errors were expressed using 2D metrics.

Razlaw *et al* [70] also compares point cloud aggregation methods. Specifically pairwise, incremental and incremental with multiresolution surfel maps, using the ICP, GICP, MGICP, Normal Distribution Transform (NDT) and surfel registration algorithms. Their study however does not present the results or data from this experiment, stating only that the incremental surfel aggregation method achieved the best results [70].

4.3 State Estimation Filters

Within the field of mobile robotics, localizing the robot is the first and arguably the most important task. If the robot does not know where it is within the environment, it cannot hope to navigate it or successfully perform any task which requires positioning. Localization often uses some form of state estimation, where the "state" is the robot's position and orientation at a given time. By far the most commonly used state estimation algorithm is the Kalman Filter (or one of its variants).

4.3.1 Kalman Filter

A Kalman Filter (KF) models the movement of the robot using two state space equations:

$$\text{motion model:} \quad \mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t \tag{4.6}$$

$$\text{observation model:} \quad \mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t \tag{4.7}$$

Where:

$\mathbf{x}_t$	=	The system state at time t, typically pose and orientation of the robot
$\mathbf{u}_t$	=	The control vector, typically motor input
$\mathbf{z}_t$	=	The measured output, as measured by any odometers, IMUs, GNSS receivers or other positioning sensors
$\mathbf{w}_t$	=	Process noise. Assumed to have a zero-mean multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{Q})$ with covariance $\mathbf{Q}$
$\mathbf{v}_t$	=	Sensor or measurement noise. Also assumed to have a zero-mean multivariate Guassian distribution $\mathcal{N}(0, \mathbf{R})$ with covariance $\mathbf{R}$
$\mathbf{A}$	=	The state transition matrix
$\mathbf{B}$	=	The control matrix
$\mathbf{H}$	=	The observation matrix

In the context of localization, the motion model represents the kinematic and dynamic behaviour of the robot and where it is expected to be at time t given the state and motor inputs at time $t-1$. The observation model represents the on-board sensors of the robot. The underlying assumption of a KF is that the state can be modelled as a Gaussian distribution $\mathcal{N}(\bar{\mathbf{x}}_t, \mathbf{C}_t)$, with a mean state $\bar{\mathbf{x}}_t$ and covariance matrix $\mathbf{C}_t$. The visual representation of this is a single point as the state mean, surrounded by an ellipse whose size is dictated by the covariance (Figure 4.4 shows a 2D example). The path that consecutive mean states plot out may or may not be the robot's true path.

By tracking just the mean and the covariance of the state, KFs rely on very little information and so can be very fast and efficient at state estimation, hence their widespread popularity.

All Kalman Filters (regardless of variant) operate by iterating through two stages: prediction and correction. In the prediction stage, the motion model is used to estimate the next state mean and covariance. In the correction stage, this estimate is compared to the measured state and updated. At this point extra notation is added. Because in reality the state and covariance are only ever estimated, they are denoted with hats: $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{C}}_t$.

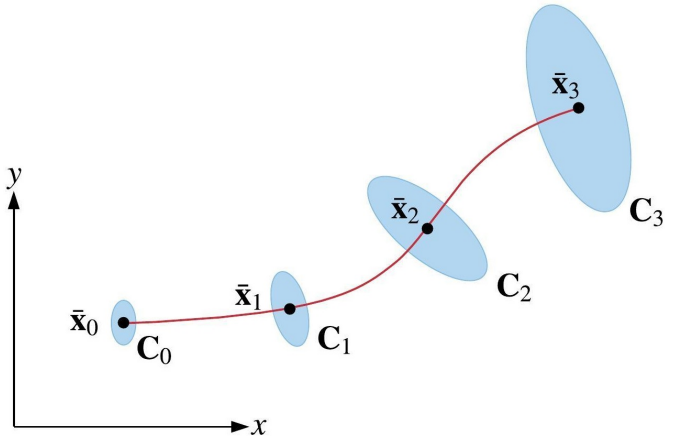


FIGURE 4.4: 2D visual representation of discrete robot poses modelled as Gaussians

Then, a $^-$ or $^+$ superscript is added to denote the mean and covariance before (*a priori*) or after (a posteriori) the update. Mathematically, this is calculated as follows:

1. **Prediction:**

$$\hat{\mathbf{x}}_t^- = \mathbf{A}\hat{\mathbf{x}}_{t-1}^+ + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t \tag{4.8}$$

$$\hat{\mathbf{C}}_t^- = \mathbf{A}\hat{\mathbf{C}}_{t-1}^+\mathbf{A}^T + \mathbf{Q} \tag{4.9}$$

   Recall that $\mathbf{Q}$ is the process noise covariance. The initial state and covariance are given as $\hat{\mathbf{x}}_0^+$ and $\hat{\mathbf{C}}_0^+$.

2. **Correction:**

   First the *innovation* is calculated, which is the difference between the measurements and the estimate:

$$\mathbf{y}_t = \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^- \tag{4.10}$$

$$\mathbf{S}_t = \mathbf{H}\hat{\mathbf{C}}_t^-\mathbf{H}^T + \mathbf{R} \tag{4.11}$$

   In Equation 4.10 the $\mathbf{H}\hat{\mathbf{x}}_t^-$ term is the *estimated* output while $\mathbf{z}_t$ is the *actual* or measured output. Recall that $\mathbf{R}$ is the sensor noise covariance. The covariance innovation is then used to calculate the optimal *Kalman gain*, which is a weighting factor that can be applied to weight the state model estimate $\hat{\mathbf{x}}_t^-$ vs the sensor measurements $\mathbf{z}_t$:

$$\begin{aligned}\mathbf{K}_t &= \hat{\mathbf{C}}_t^-\mathbf{H}^T\mathbf{S}_t^{-1} \\ &= \hat{\mathbf{C}}_t^-\mathbf{H}^T(\mathbf{H}\hat{\mathbf{C}}_t^-\mathbf{H}^T + \mathbf{R})^{-1}\end{aligned} \tag{4.12}$$

Which in turn is then used to update the state estimate and covariance:

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-) \tag{4.13}$$

$$\hat{\mathbf{C}}_t^+ = (1 - \mathbf{K}_t\mathbf{H}_t)\hat{\mathbf{C}}_t^- \tag{4.14}$$

$\hat{\mathbf{x}}_t^+$ and $\hat{\mathbf{C}}_t^+$ become the starting estimates for the next prediction and correction cycle.

The standard KF only works for linear systems, because it assumes that the state has a Gaussian distribution. If you put a Gaussian random variable (multivariate or otherwise) through a non-linear function, then the product becomes non-Gaussian and the assumption no longer holds. Because most real-world systems can be highly non-linear, several variants of the KF have been developed. The Extended and Unscented Kalman Filters (EKF and UKF respectively) are the two leading non-linear variants of the KF and are discussed in the following sections.

4.3.2 Extended Kalman Filter

The EKF works by computing a first-order linearization of the motion model around the best current estimate [96]. In an EKF, the linear requirements for the motion and observation model are dropped, and the models are instead represented by functions $f()$ and $h()$ which may differentiable:

$$\text{motion model:} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{w}_t \tag{4.15}$$

$$\text{observation model:} \quad \mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{v}_t \tag{4.16}$$

Where all other multivariate terms have the same definitions as given for Equations 4.6 and 4.7. However, in this form the motion and observation functions $f()$ and $h()$ cannot be applied directly to the state covariance matrix, so instead they are computed as matrices of partial derivatives, i.e. Jacobians $\mathbf{F}_{t-1}$ and $\mathbf{H}_t$:

$$\mathbf{F}_{t-1} = \left.\frac{\partial f(\mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{w}_t)}{\partial \mathbf{x}_{t-1}}\right|_{\hat{\mathbf{x}}_{t-1}^+, \mathbf{u}_t, 0} \tag{4.17}$$

$$\mathbf{H}_t = \left.\frac{\partial h(\mathbf{x}_t, \mathbf{v}_t)}{\partial \mathbf{x}_t}\right|_{\hat{\mathbf{x}}_t^-, 0} \tag{4.18}$$

The Jacobians are also used in the linearization of the motion and observation model, which requires computing the first two terms of a Taylor series expansion as shown:

$$f(\mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{w}_t) \approx f(\hat{\mathbf{x}}_{t-1}^{+}, \mathbf{u}_t, 0) + \mathbf{F}_{t-1}(\mathbf{x}_{t-1} - \hat{\mathbf{x}}_{t-1}^{+}) + \mathbf{w}_t{}' \tag{4.19}$$

$$h(\mathbf{x}_t, \mathbf{v}_t) \approx h(\hat{\mathbf{x}}_t^{-}, 0) + \mathbf{H}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t^{-}) + \mathbf{v}_t' \tag{4.20}$$

Where the noise terms $\mathbf{w}_t'$ and $\mathbf{v}_t'$ are also updated with Jacobians:

$$\mathbf{w}_t' = \left.\frac{\partial f(\mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{w}_t)}{\partial \mathbf{w}_t}\right|_{\hat{\mathbf{x}}_{t-1}^{+}, \mathbf{u}_t, 0} \mathbf{w}_t \tag{4.21}$$

$$\mathbf{v}_t' = \left.\frac{\partial g(\mathbf{x}_t, \mathbf{v}_t)}{\partial \mathbf{v}_t}\right|_{\hat{\mathbf{x}}_t^{-}, 0} \mathbf{v}_t \tag{4.22}$$

Note that it is typically the case that the process and sensor noise ($\mathbf{w}_t$ and $\mathbf{v}_t$) cannot be adequately measured. So common convention is to use their mean value, which is zero as per Equations 4.6 and 4.7. The updated equations for the EKF prediction and correction cycle are then as follows:

$$\text{Prediction:} \quad \hat{\mathbf{x}}_t^{-} = f(\hat{\mathbf{x}}_{t-1}^{+}, \mathbf{u}_t, 0) \tag{4.23}$$

$$\hat{\mathbf{C}}_t^{-} = \mathbf{F}_{t-1}\hat{\mathbf{C}}_{t-1}^{+}\mathbf{F}_{t-1}^{T} + \mathbf{Q}_t' \tag{4.24}$$

$$\text{Innovation:} \quad \mathbf{y}_t = \mathbf{z}_t - h(\hat{\mathbf{x}}_t^{-}, 0) \tag{4.25}$$

$$\mathbf{S}_t = \mathbf{H}\hat{\mathbf{C}}_t^{-}\mathbf{H}^{T} + \mathbf{R}_t' \tag{4.26}$$

$$\text{Kalman gain:} \quad \mathbf{K}_t = \hat{\mathbf{C}}_t^{-}\mathbf{H}^{T}\mathbf{S}_t^{-1} \tag{4.27}$$

$$\text{Correction:} \quad \hat{\mathbf{x}}_t^{+} = \hat{\mathbf{x}}_t^{-} + \mathbf{K}_t\mathbf{y}_t \tag{4.28}$$

$$\hat{\mathbf{C}}_t^{+} = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\hat{\mathbf{C}}_t^{-} \tag{4.29}$$

For a thorough derivation of these equations, readers should refer to dedicated texts such as [96] and [97]. Unfortunately, the EKF is notorious for being poor at predicting highly non-linear systems. This is due to the fact that the point of linearization is the mean estimate of the state, not the true state itself. In context, this means that if the motion of the robot is non-linear, the EKF position and orientation estimate may be biased or inconsistent [97, 98]. In addition, if the motion or observation model is incorrect, the EKF result can become increasingly inaccurate.

4.3.3 Unscented Kalman Filter

Where the EKF attempts to estimate the state of a non-linear system by linearizing the system models about the current estimate, the Unscented Kalman Filter (UKF) takes a deterministic approach called the *Unscented Transform* (UT). The UT selects a number of sample points to represent the mean and covariance of the state, and then propagates them through the system models. This uses the true non-linear models rather than a linearized approximation, and achieves 3rd order accuracy as opposed to the EKFs 1st order [99]. The UT is based on the assumption that *"it is easier to approximate a probability distribution than it is to approximate an arbitrary nonlinear function or transformation"* [100].

Simply put, the UT chooses a set of $2n+1$ sigma points $\mathcal{X}_i$ such that they approximate the mean and co-variance of the state ($\bar{\mathbf{x}}$ and $\mathbf{C}$) where n is the dimensionality of the state. The sigma points are defined with corresponding scalar weights as follows:

$$\mathcal{X}_0 = \bar{\mathbf{x}} \qquad w_0 = \kappa/(n+\kappa) \tag{4.30}$$

$$\mathcal{X}_i = \bar{\mathbf{x}} + \left(\sqrt{(n+\kappa)\mathbf{C}}\right)_i \qquad w_i = 1/2(n+\kappa) \qquad i = 1, ..., n \tag{4.31}$$

$$\mathcal{X}_{i+n} = \bar{\mathbf{x}} - \left(\sqrt{(n+\kappa)\mathbf{C}}\right)_i \qquad w_{i+n} = 1/2(n+\kappa) \qquad i = n+1, ..., 2n \tag{4.32}$$

Where κ is a tuning parameter. The definition of the weights (w_i) varies between researchers, and some such as [99] define different weights for the state and covariance. For simplicity, Equations 4.30 to 4.32 use the original definition provided by Julier [101] and apply the same weights to both state and covariance calculations. Julier also recommends that the value of κ be chosen such that $n + \kappa = 3$ when the state is assumed to be Gaussian.

Having chosen appropriate sigma points and weights, the points themselves can be propagated through the non-linear motion model $f()$ from Equation 4.15:

$$\mathcal{X}_{t,i} = f(\mathcal{X}_{t-1,i}, \mathbf{u}_t) \tag{4.33}$$

The propagated sigma points $\mathcal{X}_{t,i}$ are then weighted and used to construct the *a priori* state and covariance estimate:

$$\hat{\mathbf{x}}_t^- = \sum_{i=0}^{2n} w_i \mathcal{X}_{t,i} \tag{4.34}$$

$$\hat{\mathbf{C}}_t^- = \sum_{i=0}^{2n} w_i (\mathcal{X}_{t,i} - \hat{\mathbf{x}}_t^-)(\mathcal{X}_{t,i} - \hat{\mathbf{x}}_t^-)^T \tag{4.35}$$

This is the key to the UKF, as the *a priori* estimates calculated using propagated sigma points more accurately represent the state mean and covariance than the linearization computed in an EKF. The remainder of the Kalman prediction-correction follows the same structure, leading to revised equations shown below:

Prediction:

$$\mathcal{X}_{t,i} = f(\mathcal{X}_{t-1,i}, \mathbf{u}_t) \tag{4.36}$$

$$\hat{\mathbf{x}}_t^- = \sum_{i=0}^{2n} w_i \mathcal{X}_{t,i} \tag{4.37}$$

$$\hat{\mathbf{C}}_t^- = \sum_{i=0}^{2n} w_i (\mathcal{X}_{t,i} - \hat{\mathbf{x}}_t^-)(\mathcal{X}_{t,i} - \hat{\mathbf{x}}_t^-)^T \tag{4.38}$$

$$\mathcal{Z}_{t,i} = h(\mathcal{X}_{t,i}) \tag{4.39}$$

$$\hat{\mathbf{z}}_t^- = \sum_{i=0}^{2n} w_i \mathcal{Z}_{t,i} \tag{4.40}$$

Innovation:

$$\mathbf{y}_t = \mathbf{z}_t - \hat{\mathbf{z}}_t^- \tag{4.41}$$

$$\mathbf{C}_{yy,t} = \sum_{i=0}^{2n} w_i (\mathcal{Z}_{t,i} - \hat{\mathbf{z}}_t^-)(\mathcal{Z}_{t,i} - \hat{\mathbf{z}}_t^-)^T \tag{4.42}$$

$$\mathbf{C}_{xy,t} = \sum_{i=0}^{2n} w_i (\mathcal{X}_{t,i} - \hat{\mathbf{x}}_t^-)(\mathcal{Z}_{t,i} - \hat{\mathbf{z}}_t^-)^T \tag{4.43}$$

Kalman gain:

$$\mathbf{K}_t = \mathbf{C}_{xy,t}\mathbf{C}_{yy,t}^{-1} \tag{4.44}$$

Correction:

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t \mathbf{y}_t \tag{4.45}$$

$$\hat{\mathbf{C}}_t^+ = \hat{\mathbf{C}}_t^- - \mathbf{K}_t \tag{4.46}$$

In addition to being more accurate at state estimation in general, the UKF has a few additional benefits. It does not require computing the Jacobians of the motion or observation model, and there is no requirement that the models be smooth and differentiable [97]. The UKF is also more accurate than the EKF when used in SLAM systems [102].

4.4 Simultaneous Localization and Mapping

The term "Simultaneous Localization and Mapping" (SLAM) is very broad, and refers to any process which builds a map around an observer while at the same time localizing the observer within it. Because of the size and complexity of this problem, there is a huge variety of different approaches to solving it. Most belong to one of three basic categories, which are defined by the underlying algorithm: an Extended Kalman Filter (EKF SLAM), particle filter (Fast SLAM) or pose graph (Graph SLAM).

The rest of this section will briefly discuss each of these three categories of SLAM algorithms, as well as some of the pros and cons of each. SLAM is a widely studied topic, and the details are best discussed in a dedicated text. Readers wishing to learn more about SLAM can refer to *Probabilistic Robotics* by Thrun *et al.* [96], *Springer Handbook of Robotics* by Siciliano and Khatib [103], or *Robotics Vision and Control* by Corke [104].

4.4.1 EKF SLAM

Early solutions to the SLAM problem were based on the Extended Kalman Filter discussed in Section 4.3.2. Where the mean state estimate $\bar{\mathbf{x}}$ tracks the current pose of the robot $\mathbf{x}_t$ and the map $\mathbf{m}$, which consists of any observed features (also called landmarks l_i) in the environment [105, 106]. The new state and covariance matrix are defined as follows:

$$\bar{\mathbf{x}}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_t \\ l_1 \\ \vdots \\ l_n \end{bmatrix} \tag{4.47}$$

$$\mathbf{C}_t = \begin{bmatrix} \mathbf{C_{xx}} & \mathbf{C_{xm}} \\ \mathbf{C_{mx}} & \mathbf{C_{mm}} \end{bmatrix} \tag{4.48}$$

As with the stand-alone EKF algorithm, EKF SLAM models the state and covariance as a multivariate Gaussian $\mathcal{N}(\bar{\mathbf{x}}_t, \mathbf{C}_t)$, and iterates through the same predict-correct cycle. Because it only stores the current location of the robot, it is referred to as an *online* SLAM solution [96].

This approach utilizes an abstracted landmark-based model of the environment, as opposed to the map representations previously discussed in Section 4.1. Every new landmark adds a new row to the state estimate, while adding a new row *and* column to the covariance. As the covariance grows quadratically, the next state and covariance estimate becomes progressively more expensive to compute [103]. This scales poorly for large maps that contain more than a few hundred landmarks [107]. Researchers have attempted to solve this problem by breaking the map into smaller sub-maps [108–113].

In addition, EKF SLAM is sensitive to motion and measurement noise, which cause uncertainty in the position of landmarks or the robot itself [114]. This can cause the EKF algorithm to diverge, especially if some landmarks are misidentified [114, 115]. After being fused in the next state estimate, any mistakes like this cannot be later corrected.

4.4.2 Particle Filter SLAM

The next major category of SLAM solutions to be developed are based on Particle Filters. In the context of localization, particle filters are Monte Carlo methods which approximate the position of the robot not as a single state $\mathbf{x}_t$ and covariance, but as a set of n possible states $\{\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, ..., \mathbf{x}_{t,n}\}$ (called particles or samples) [103, 107]. Each particle contains an estimate of the current state, and a Gaussian representing each landmark in the map. Particles are updated individually by the motion model, given a set of control inputs and observations. The true state of the robot is then represented by the *distribution* of particles. I.e. the best estimate of the true state is the region of state space where the density of particles is highest [96].

These types of SLAM algorithms are sometime referred to as Fast SLAM. They are also sometimes called Rao-Blackwellized Particle Filter SLAM, named after the Rao-Blackwell factorization technique that allows them to efficiently compute the robot's trajectory and map.

The particles are initialized by setting their state to the starting location of the robot. Then, much like an EKF, the Fast SLAM particle filter iterates through a prediction and correction step, which includes re-sampling the particles. This occurs as follows:

1. **Prediction:**

   The control input $\mathbf{u}_t$ is used, along with the motion noise $\mathbf{w}_{t,i}$, to project each particle prior $\hat{\mathbf{x}}^+_{t-1,i}$ through the motion model. The posterior is then that particle's prediction of where the robot will be at the next time step [96]:

$$\hat{\mathbf{x}}^-_{t,i} = f(\hat{\mathbf{x}}^+_{t-1,i}, \mathbf{u}_t, \mathbf{w}_{t,i}) \tag{4.49}$$

2. **Correction/re-sampling:**

   The expected sensor output $\hat{\mathbf{z}}_{t,i}$ is calculated by inputting the newly calculated posterior to the observation model [96]:

$$\hat{\mathbf{z}}_{t,i} = h(\hat{\mathbf{x}}^-_{t,i}, 0) \tag{4.50}$$

   And then compared to the measured sensor output $\mathbf{z}_{t,i}$. The divergence between them is used to generate an *importance* weight $w_{t,i}$ for each particle, which are normalized such that they sum to 1. This highly weights particles whose prediction was close to the measured sensor output, and de-weights those that were not [107]. A new set of particles are drawn with replacement from the existing set (i.e. resampled), where the importance weight is used to determine which particles are selected and which are discarded.

Compared to EKF SLAM, Fast SLAM does not suffer the same growth in complexity, as each particle grows linearly with each new landmark, rather than quadratically [103, 107, 116]. Another advantage is that Fast SLAM can model non-Gaussian, non-linear systems [117].

There are some disadvantages to Fast SLAM. One being that the importance of the number of particles varies significantly depending on the environment [103]. Researchers have demonstrated that recent iterations of Fast SLAM can be successful with a small number of particles, even as few as one [116, 118]. However most research shows that a large number of particles in the tens or even hundreds gives better performance [116, 118, 119]. But increasing the number of samples increases computation time so it cannot be set to an arbitrarily large value [120]. Outside of empirical analysis, most choices for this value are an educated guess.

Another disadvantage is that like EKF SLAM, many Fast SLAM implementations depend on accurately observed landmarks. Noise in motion or observations measurements can lead to misidentified landmarks, causing particles to diverge from the true state.

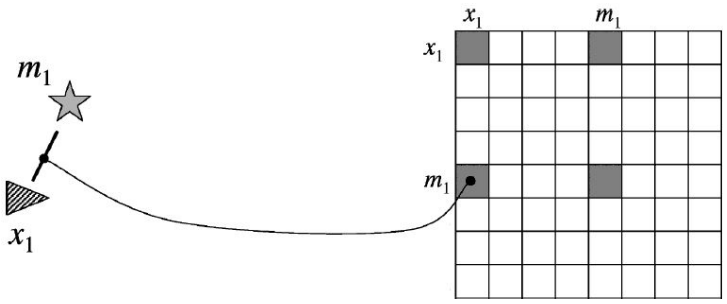
4.4.3 Graph SLAM

As the robot moves, adjacent poses are linked by a *motion constraint*, using the robot's motion model and noise covariance. Every feature is likewise linked to every pose at which it was observed by a *measurement constraint*, using the robot's sensor measurement function and noise. These constraints are used to construct a graph which is referred to as a *pose graph*. Each constraint is analogous to a spring that can be stretched or squashed to find the most likely position of each robot pose and feature. A least-squares solution can then be formulated to solve for the optimal arrangement which minimizes the sum of the motion and measurement constraints [96]. The basic Graph SLAM algorithm, as described in [96], does this in three steps:

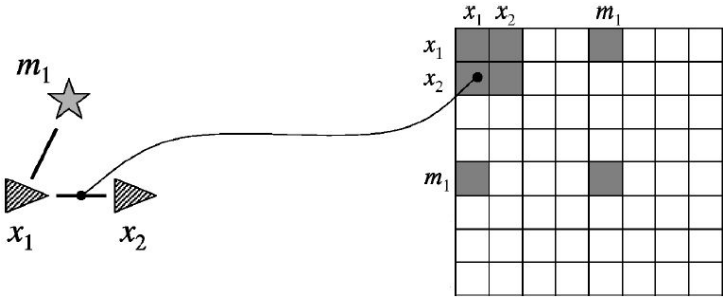
1. **Linearize:** Each motion and measurement constraint is linearized using a Taylor expansion and entered in an information matrix, as illustrated in Figure 4.5 with four example poses.
2. **Reduce:** One-by-one, each feature is removed and all it's measurement constraints are turned into new motion constraints between all the poses that observed that feature. This removes the map from the information matrix and reduces it's size.
3. **Solve:** The pose graph is solved in a least-squares formulation to find the optimal position of each pose.

An important characteristic of Graph SLAM is that because the graph stores *every* robot pose and feature, no information is lost. So when solving for the map and robot path, the graph can be iterated over multiple times to improve the map and path estimate. This can aid the algorithm during loop closure. This also means that the *entire* robot path is computed, not just it's current position. As a result, graph-based SLAM algorithms generally produce more accurate maps than other SLAM algorithms [121].

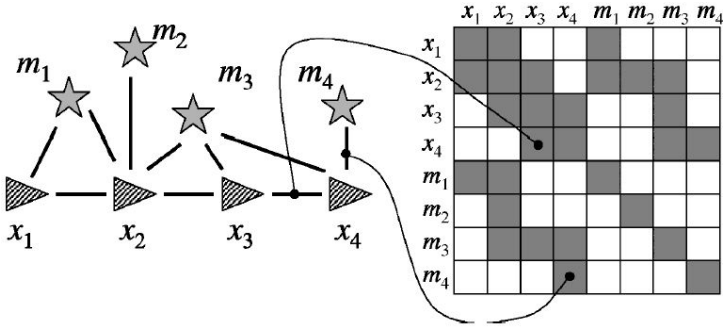
However Graph SLAM is not without flaws, as the robot moves through it's environment the information matrix can become prohibitively large. Variations of Graph SLAM work



(a) The feature m_1 is linked to the pose at which it was observed (x_1) by a measurement constraint, which is entered in an information matrix (right).



(b) Adjacent poses such as x_1 and x_2 are linked by motion constraints, also entered in the information matrix



(c) A new row and column is added for every new pose or feature, producing the sparse matrix shown here.

FIGURE 4.5: Illustration of how Graph SLAM builds an information matrix to represent the robot's motion and environment

Source: Thrun, Burgard and Fox, 2005 [96]

around this by dividing the map into smaller sub-maps [122, 123] or by limiting the number of robot poses [124]. Large maps can also make loop closure harder, potentially resulting in mis-aligned features within the map. In theses cases corrective algorithms such as the Lazy Data Association (LDA) employed in [68] can improve map quality.

4.5 Flaws in Existing SLAM Methodologies

What should become apparent from the previous section, and other SLAM literature, is that the focus of SLAM development has been on achieving a real-time solution for estimating a robot's position. The accurate positioning of landmarks and features is only a means to achieving this end.

And yet SLAM is increasingly being used to generate volumetric rather than feature-based maps, with the express purpose of mapping the environment. For algorithms like EKF SLAM or Fast SLAM, this requires extending their basic implementations to incorporate depth or LiDAR information. But these extensions do not change the underlying algorithms, and there are several underlying design principles that mean these algorithms are not conducive to accurate mapping.

As research into SLAM has progressed, many research teams are increasingly making their implementation available on the internet as open-source software. Table 4.1 lists some of the more popular packages, and the core SLAM algorithm they are based on.

TABLE 4.1: Open-Source SLAM packages

Name	SLAM type	Citation
Google Cartographer	Graph	[66]
LiDAR Odometry and Mapping (LOAM)	Scan	[69, 88]
Berkley Localization and Mapping (BLAM)	Graph	[125]
Real-Time SLAM (RT-SLAM)	EKF	[126, 127]
Real-Time Appearanced-Based Mapping (RTAB-Map)	Graph	[128]
ORB-SLAM2	Graph	[129, 130]
Hector SLAM	EKF	[131]
Gmapping	Particle	[132]

With reference to some of the packages, the remainder of this section will explain some of the inherent flaws that can affect the quality of a map produced by a SLAM system.

Use of Odometry only as Initial Estimate

Many SLAM algorithms are built on the principle that odometry information will either not be available (SLAM is virtually always assumed to operate in GNSS-denied environments), or that it will always be inaccurate. So this information is typically only used an an estimate for where the next pose of the robot is, and is later overridden by observations from visual sensors. For this same reason, virtually all SLAM systems do

not explicitly allow for the input of GNSS position data, treating any input odometry data as relative motion, not absolute.

For example, Google Cartographer only uses the odometry and IMU data to position the next LiDAR scan. Once this scan has been registered to the previous scan, the post-registration pose overrides the pose suggested by the odometry. This is the case even if the odometry information was more accurate, or if the scan registration failed.

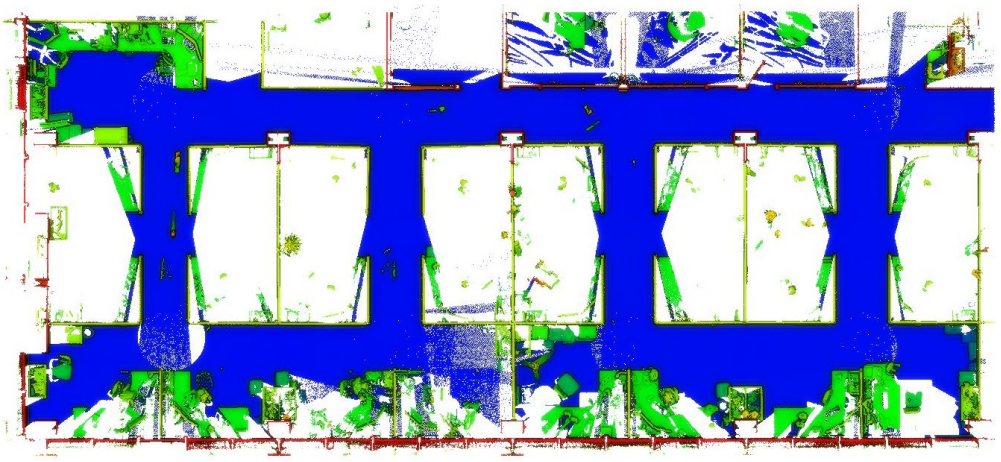
Lack of Failure Containment

The focus on achieving real-time performance with SLAM has led to many algorithms which either operate online (i.e. they only retain the latest robot pose) or lock observed data into sub-maps which cannot later be changed. Not only does this throw away past poses and observations, but also any uncertainty information (like covariance matrices) that might have indicated how accurate those past observations were.

The end result is that when errors occur, they are often not detected and become locked into the map, thus permanently affecting the robot's trajectory or the shape of the map. Figure 4.6 shows an example where subsequent LiDAR scans have been matched with an angular offset in LOAM, causing the remaining map and trajectory to be skewed.

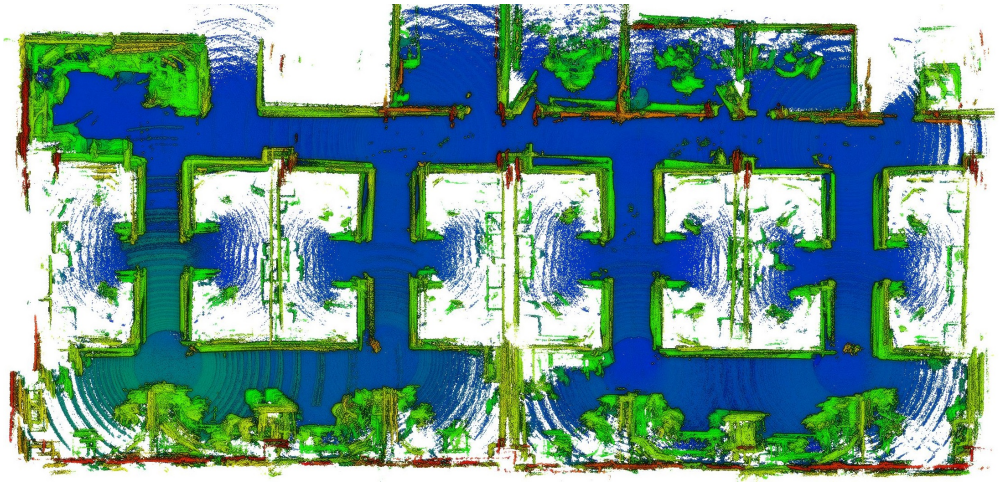
Figure 4.6(b) shows an extreme case caused by a gross error in point cloud registration. However this can also occur over long trajectories due to small incremental errors. For example, Figure 4.6(c) shows a result of the same area from Google Cartographer. The cloud is coloured by height, and readers will observe that the ground in top-left of the cloud (where the robots trajectory started) is blue, while the bottom-left corner (where the trajectory ended) is blue-green - indicating a difference in height. This has occurred because of a systematic upwards bias in the odometry or point cloud registration. This is particularly noticeable in maps spread across multiple floors of a single building, such as the LOAM output shown in [113].

This is a particular problem in SLAM systems that build sub-maps because most packages that do this (such as Cartographer) lock the data within the sub-map. So although the positions of each sub-map can be later corrected by pose graph optimization, the errors within them cannot be. As a result, these errors can irreparably warp the map and robot trajectory. This may not be a concern if the map is only intended for navigation purposes. But for accurate mapping, these maps can be highly unsuitable.



(a) SX10 ground truth point cloud scan of office cubicles

(b) Downsampled LOAM output showing a rotation error in the map



(c) Google Cartographer output, coloured by height. This shows a systematic upwards bias in the algorithm

FIGURE 4.6: Example of errors in a SLAM map.

Loop closure is a technique employed specifically to counter this problem. But not all SLAM systems use it (e.g. LOAM) and especially bad errors can cause algorithms which do not explicitly look for closed loops to fail to detect them.

4.6 Summary of Robotic Mapping

Creating a map with a mobile robot is a difficult and complicated problem that requires a variety of different algorithms to solve. Multiple algorithms are typically combined in a Simultaneous Localization and Mapping (SLAM) system. Often these combined systems are extensions of localization algorithms such as the Kalman or Particle filters.

Because the primary focus of SLAM development has been on achieving real-time positioning of the robot for navigation purposes, most historical SLAM systems use abstract feature-based maps which are unsuitable for geospatial surveying. In recent years, more LiDAR-based SLAM systems have been developed, but problems such as poor cloud registration or Loss of Significance can introduce errors in the robot or cloud pose. Some of the SLAM design principles that are inherited for historical reasons, or required for real-time performance, can lock these errors into the map in a way that becomes difficult or impossible to correct after the fact.

In addition, the feature-based approach to cloud registration or map building can fail in areas that don't have many well-defined features. This is why SLAM systems are typically showcase in urban or indoor environments. Many areas that surveyors would want to make with an autonomous mobile robot are outdoors and contain many unstructured features such as trees or bushes.

For all of these reasons, SLAM systems are often designed in ways that are not conducive to accurate, reliable geospatial mapping. If an autonomous mobile robot is to be used to create a map that can be compared to that produced by conventional survey instruments (to answer the Research Questions in Section 1.2), then it must be designed for-purpose. The mapping methodology that the robot uses must also exploit the availability of GNSS data and explicitly search for errors in the map as it is being built and handle them appropriately.

5 | Design of Proposed Autonomous Mapping Robot

This chapter details the development of the autonomous robot: the design process, how it was built, and how it was programmed to achieve the desired functionality. Following the Research Questions, the initial objective was to develop a very basic autonomous surveying system using open-source software and off-the-shelf hardware. This prototype was later expanded to include a LiDAR unit for mapping purposes.

Some stages of development, such as selecting the hardware and operating system for the robot, required a straight-forward choice between several existing option. For these stages, this chapter will briefly discuss the options that were available, what was chosen, and why. Other stages, such as achieving autonomy, required writing software specifically for a task or using existing software packages and modifying configuration files appropriately.

This chapter provides a high-level overview of how the robot operates. Specific implementation details are provided in Appendix B.

5.1 Hardware

This section briefly describes the chassis and sensors what were chosen for the robotic platform and why. The chassis is discussed first as they often include inbuilt sensors, whose type and quality affect the need for additional sensors.

5.1.1 Chassis

The selection of the chassis was made prior to commencement of the project. The sponsor of this research, Trimble Inc, has an existing relationship with Clearpath Robotics, a Canada-based manufacturer of robotics platforms. Clearpath Robotics produces a variety of ground and water based robotic platforms, all of which are designed with autonomous control in mind. Clearpath's products are often used in academic research and some of these platforms are already in use within Trimble. For this reason a Clearpath "Jackal" UGV platform was made available for this research. Figure 5.1 illustrates the shape and dimensions of the Jackal chassis.

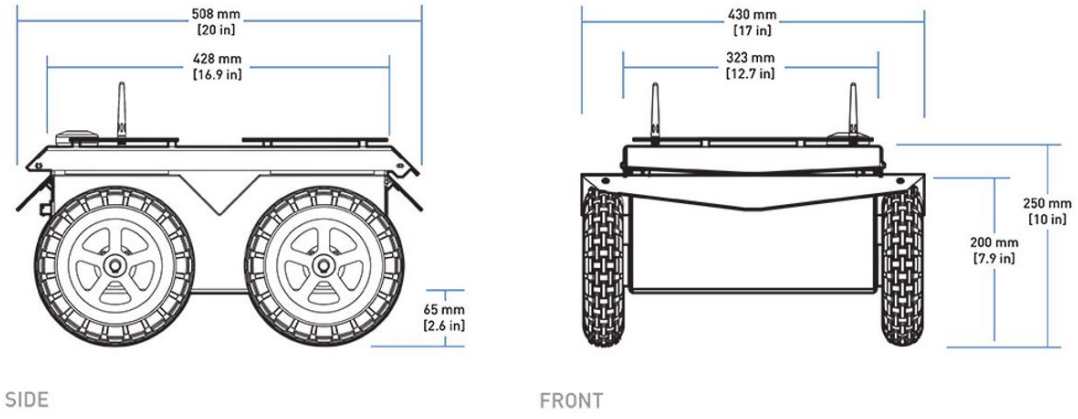


FIGURE 5.1: Dimensions of the Clearpath "Jackal" chassis

Source: Clearpath Robotics, 2016 [133]

The data sheet for Clearpath Jackal can be found on the Clearpath Robotics website. It arrived pre-configured with Ubuntu Server 14.04, the Robot Operating System (ROS) distribution Indigo Igloo and the core ROS drivers necessary for ROS to control the Jackal. Information on modern versions of the same robot can be found on the Clearpath Robotics website. Source code for the Clearpath ROS drivers can be found on their Github repository [134].

5.1.2 Sensors

The Jackal chassis ships with several on-board sensors: an InvenSense MPU-9250A Inertial Measurement Unit (IMU), quadrature encoders on each motor, and a Garmin GPS receiver. While both the IMU and the encoders were deemed to be sufficient, the GPS was not because it was only accurate to 1-3 meters. A survey-grade GNSS system must be accurate to 1-2 *centimeters*, so several Trimble receivers were researched as an alternative. Trimble R7, R10 and SPS855 GNSS receivers were all considered as potential upgrades for the UGV. The R7 GNSS receiver was chosen for the following three reasons:

1. **Accuracy** - In Real Time Kinematic (RTK) mode the R7 is accurate to 15 mm RMSE in the vertical plane and 8 mm RMSE in the horizontal plane.
2. **Modularity** - The R7 uses an external antenna, which can be easily mounted on a mast elsewhere on the robot, thereby improving reception.
3. **Simplicity** - The R7 does not require bi-directional communication and can be configured to output generic NMEA GPS messages via the serial port. This makes integrating the R7 with other devices much easier.

A Trimble Zephyr 2 antenna was selected to supplement the R7 receiver, and both were firmly mounted on the chassis' mounting plates (see Figure 5.2).

The first prototype design used only the sensors described thus far, and was later upgraded with a LiDAR unit. For this, a single Velodyne HDL-32e was chosen because it was believed to have sufficient accuracy and field of view, and because a spare unit could be sourced internal to Trimble at no additional cost to the project. Later in the project, the in-built IMU was replaced with a LORD MicroStrain 3DM-GX5-25 IMU as the inbuilt IMU failed.

The designs for the complete first and second prototypes are provided the subsequent section.

5.1.3 First Prototype Design

The first prototype was designed with only the sensors necessary to localize itself in earth-referenced coordinates, with the intention of only performing the work described in Chapter 6. This prototype is shown in Figure 5.2.

FIGURE 5.2: First robot prototype

To clarify, it's sensors include an InvenSense MPU-9250A IMU, wheel encoders, Garmin GPS (not used), and Trimble R7 GNSS receiver combined with a Trimble Zephyr 2 antenna.

Note that the choice not to include visual sensors was deliberate, as it was believed that this would unnecessarily complicate the design and add time to the project. As such, this version of the UGV used a more basic system layout, which identical to the system diagram shown later in Figure 5.4 but without the LiDAR and SLAM components.

The rest of this chapter will describe the second prototype, as this is the version of the robot effectively used the same localization and navigation system, but without the use of LiDAR for mapping and obstacle avoidance.

5.1.4 Second Prototype Design

The second prototype of the mobile robot required the addition of a Velodyne HDL-32e LiDAR scanner, which needed to be mounted high on the chassis to give it a clear vertical field of view. However this clashed with the identical need of the Zephyr antenna to be mounted high in order to clear any obstacles on the robot or nearby objects which could obstruct GNSS signals.

To facilitate both of these requirements, a custom frame was made from extruded aluminum struts to which the LiDAR scanner and antenna could be mounted (see Figure 5.3).

FIGURE 5.3: Second robot prototype

This revision included an angled LiDAR mount for the HDL-32e and a mount for the LORD MicroStrain 3DM-GX5-25 IMU. To prevent hard-iron interference to the IMU from the chassis, several cables near the IMU were re-routed and nearby ferrous screws were replaced with brass non-ferrous equivalents.

5.2 Software

Most robots used for academic research are operated by a generalized development framework, rather than a optimized suite of proprietary software. Such frameworks are often open-source and provide a range of software for performing different tasks. This section describes the robotic frameworks that were considered for this research and which was chosen, as well as the general design process that was followed when developing the UGV's functionality.

5.2.1 Robot Development Framework

Many open-source robot development frameworks exist, with different design philosophies, strengths, weakness and intended applications. Current examples include Player, Robot Operating System (ROS) and Microsoft Robotics Developer Studio (MRDS). Each of these platforms was briefly researched to determine which was most suitable for this application.

The Player Project is an open-source hardware abstraction layer for robots. It encompasses the Player network server and the Stage simulator. Player implements a TCP socket-based client/server network which uses a set of standardized messages called *interfaces* to control and interact with hardware. The server side runs the hardware drivers, while the client side runs control programs and interfaces with the user. Control programs are provided by Player client libraries or the open-source community. Player supports predominantly ground-based mobile robots.

The Robotic Operating System (ROS) is an open-source robot ecosystem that consists of development frameworks, software libraries and visualization plugins. The ecosystem is built around a publisher/subscriber model of networked *nodes*. Each ROS node is an individual program or process that computes data and transfers it via *topics*, independent of any other nodes using the same data. The data itself is contained in pre-defined formats called *messages*. Nodes are provided in ROS *packages*, which are compilations of nodes, libraries, and configuration files. ROS is widely used by both industry and academia for robot research and development. ROS is designed to be platform-independent and can be run on a wide range of robots from industrial manipulators to self-driving cars.

Microsoft Robotics Developer Studio (MRDS) is a robotic control and simulation environment. It includes tools for visual programming, web or windows-based interfaces and 3D simulation. Like Player and ROS, MRDS communicates using messages with predefined formats. The core of MRDS is the Decentralized Software Service (DSS) module, a .NET-based runtime environment which manages services and applications running on the robot. MRDS is used on several ground-based robots, however it has limited support as Microsoft no longer updates or contributes to its code base.

After consideration, ROS was chosen as the preferred development platform for the following reasons:

1. The Jackal UGV ships pre-configured with ROS, and Clearpath provides tutorials and source code for basic operation. This reduces setup time and complexity.
2. Of the options researched, ROS has the largest code base and range of libraries.
3. Of the options researched, ROS has the largest on-line community which can be critical for debugging open-source code. This includes the dedicated Q&A forum ROS Answers.
4. ROS is actively supported by many commercial and non-profit robot developers such as Willow Garage, Clearpath Robotics, Rethink Robotics, Robotnik and the Open Source Robotics Foundation (OSRF).

5.2.2 Software Architecture Overview

The ROS ecosystem already has many packages for building some of these components. But even with ROS, there is no complete solution for autonomous navigation, and robotic applications even slightly outside the norm still require bespoke solutions. With the focus of this research on the mapping and localization aspects of the robot, it was desired that as many other components of the robot as possible be treated as black-box solutions. I.e. a working solution that can just be implemented and used without modification.

The localization component must create a world map from sensor data and locate the robot within the map. This requires calculating both the position and orientation of the robot (often called the robot *state*). Localization in mobile robotics is typically achieved with sensor fusion and probabilistic algorithms, such as the Kalman or Particle Filter

[135]. Sensor fusion involves mostly non-visual sensors and may be used with, or in place of, Simultaneous Localization And Mapping (SLAM). The navigation component must then plan and control the movement of the robot in the world. The core components must be managed by a over-arching layer that takes commands from the user and executes the necessary functions. Some or all of the management layer may operate on a remote device such as laptop or tablet.

For localization, the `robot_localization` package was chosen as it allows easy implementation and configuration of either an Extended or Unscented Kalman Filter. For navigation, the ROS navigation stack was used. In particular, the `navfn` and `base_local_planner` packages were desired for their path-planning capabilities, along with the `move_base` package which uses them to enable autonomous navigation. In addition, the `actionlib` package was also considered as it provided support for multiple movement goals and feedback on goal progress. Section B.2 describes these packages in greater detail and how they were configured and put together to provide autonomous navigation.

After researching and considering the available packages in the ROS ecosystem, a software architecture for the first prototype was designed and later expanded for the second prototype. The final system diagram is shown in Figure 5.4. Specific details of how the software packages were integrated and configured are provided in Appendix B.

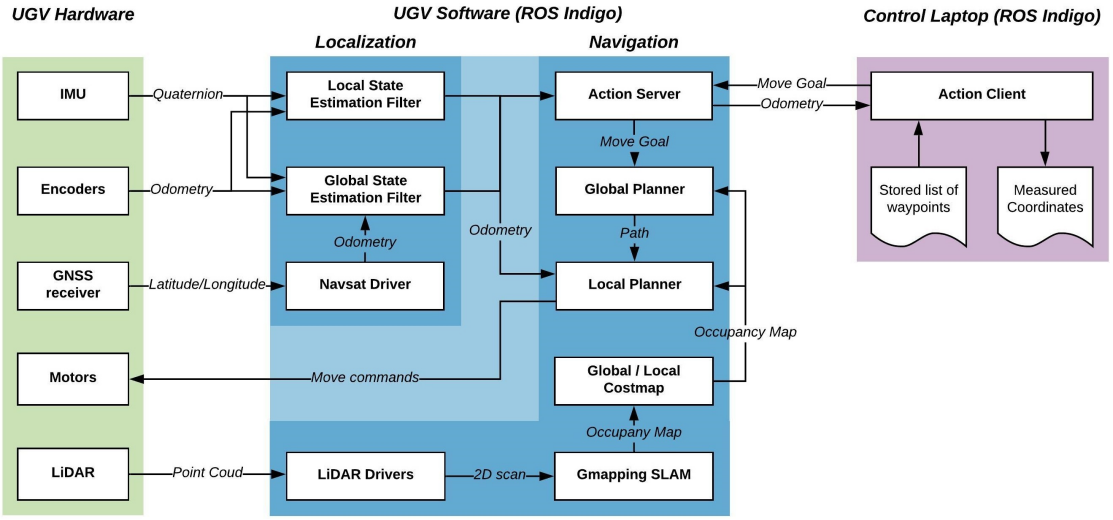


FIGURE 5.4: System diagram of robotic platform. The first prototype also used this system, without the LiDAR, LiDAR Drivers and GMapping SLAM components

5.3 Robot GNSS Calibration

Calibrating the reference frames of the robot and identifying sources of error are important for producing accurate results. The key reference frames were the frames representing the position of the robot's chassis, Zephyr 2 antenna, and Velodyne HDL-32 LiDAR unit. These distances where checked several times to confirm that measured values were within 1-2 mm of the ideal values.

The robot was also checked against several ground control points which had been established using a Trimble R10 GNSS system [2] (shown in Figure 5.5) in RTK mode. This was done to check the accuracy of the R7 and Zephry 2 GNSS setup on the robot, and check for possible sources of error such as incorrect selection of the geoid or ellipsoid model.

Each control point was observed for 3 minutes on two different days. The observation time was set to 3 minutes as this is long enough to account for instantaneous variations in the reported elevation. This would then account for most sources of error aside from environmental effects such as changes in satellite constellation. Note that in Figure 5.5, the survey pole used is 2 m tall, to elevate the antenna built into the R10 receiver and mitigate multipath reflections from the ground.

The robot was then used to survey the same points for 3 minutes directly afterwards to minimize errors due to changes in satellite constellation. From this data, the mean error (ME), root-mean-square error (RMSE), and standard deviation (SD) were extracted, using Equations 3.1 to 3.3. The raw elevation data is shown in Figure 5.6 while the error values are provided in Table 5.1, where each data set is given a number which denotes the control point it was taken at, and a letter ('a' or 'b') to denote which day it was collected.

The first calibration point was in a carpark, within a few meters of cars and the side of a building, whereas the other three control points were much further from such obstacles. Inspection of Figure 5.6(a) shows that the data is more erratic and has a wider range than the data collected at the other two control points. This is confirmed by a SD of 18.9 mm, more than double the SD values for the other data sets.

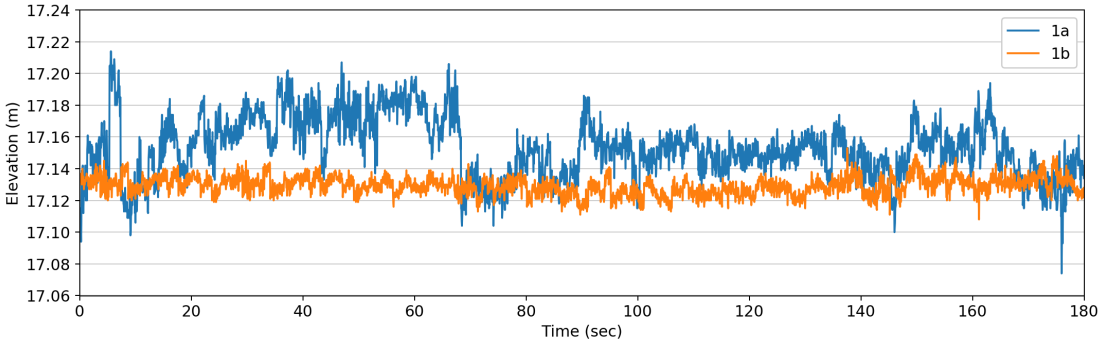
FIGURE 5.5: Example control point used to calibrate robot

The unusual error is mostly likely caused by multipath error. This occurs when GNSS signals bounce off reflective surfaces (such as the nearby cars or building) before reaching the antenna, as discussed in Section 3.4.2. It is highly dependent on the constellation of the satellites in the sky [44], and the angle of incidence of their signals with nearby objects. Hence why data set 1a appears to be strongly affected by multipath, but data set 1b (collected on the next day) does not.

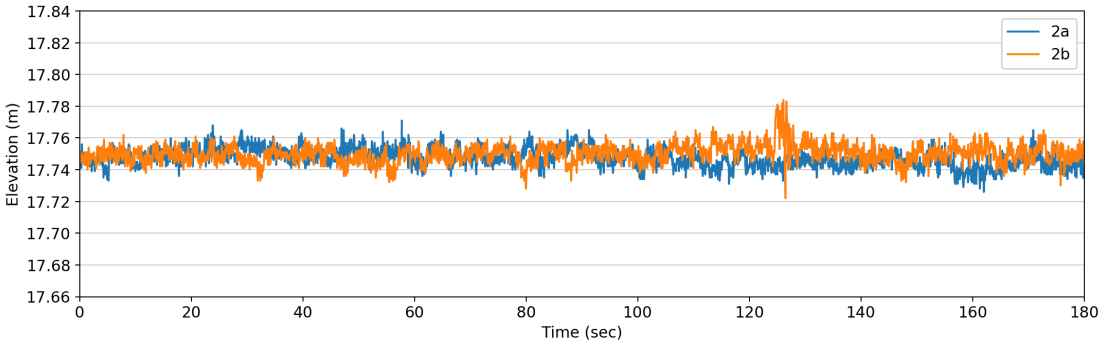
TABLE 5.1: Error statistics for robot elevation data measured at control points

Control point	R10 Elevation (m)	Robot mean Elevation (m)	ME (mm)	RMSE (mm)	SD (mm)
1a	16.423	16.446	22.8	29.7	19.1
1b	16.448	16.423	-26.1	26.7	5.8
2a	17.041	17.040	-0.9	6.0	5.9
2b	17.052	17.043	-9.2	11.0	5.9
3a	17.117	17.139	22.2	23.9	8.7
3b	17.115	17.120	4.7	7.3	5.6
4a	17.461	17.461	10.6	12.4	6.8
4b	17.477	17.472	-4.5	7.3	5.7

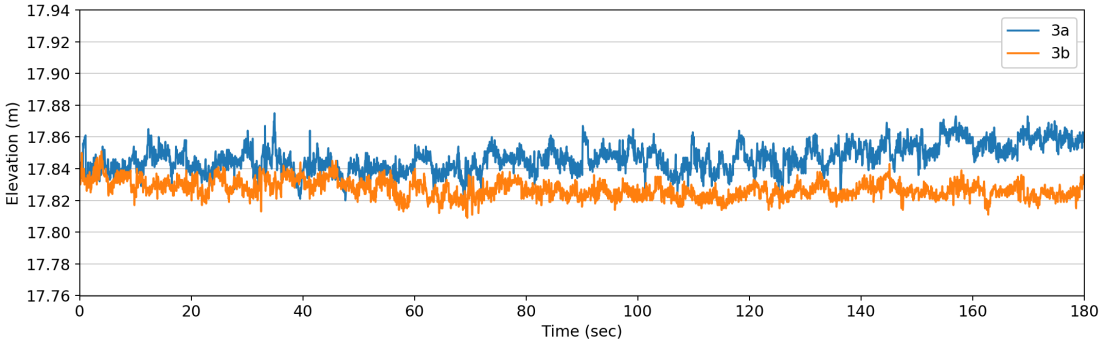
Note: every control point was measured twice, on different days. The 'a' and 'b' suffix denotes the day.



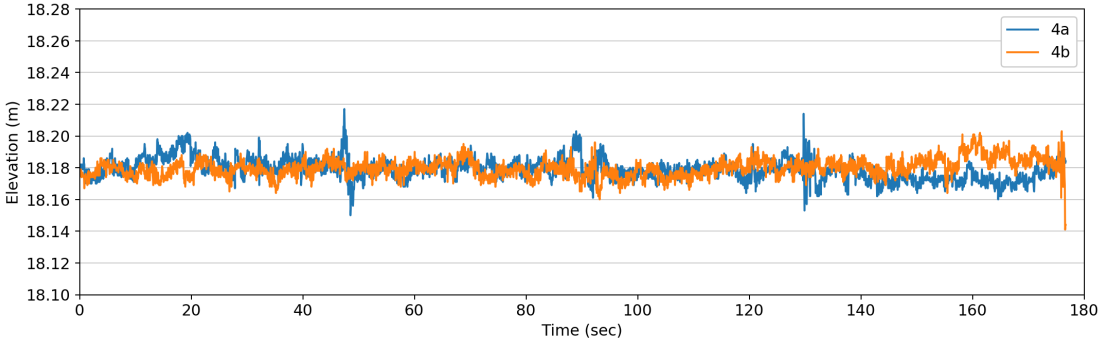
(a) Control point 1



(b) Control point 2



(c) Control point 3



(d) Control point 4

FIGURE 5.6: Elevation Data collected by robot on four control points

In observing the results in Table 5.1, the first thing to note is that the elevation for each control point is slightly different between days, regardless of whether inspecting the R10 or robot elevations. In the data shown, this difference is approximately 10-20 mm. Given that only changing variables were the time of day (i.e. changes in satellite constellation) and weather, it can be assumed that these were responsible for the error. But because each row of data in Table 5.1 was collected within a 1-2 hour window, these environmental effects should not be responsible for any difference between the R10 and robot elevation measurements.

The data collected at control point 1 was affected by multipath, and has larger ME, RMSE and SD error values as a result. So it is not representative of the robotic system's potential accuracy, but serves as an example of the level of additional error that can be expected when operating near buildings.

Ignoring the data from control point 1, the remaining results largely show an RMSE of less than 15 mm, less than the vertical error rated by the R7. The SD values are all approximately 6 mm, indicating that the measurements are relatively consistent with the mean elevation. The ME for these results fluctuates around 0 mm, producing several positive and negative values. This indicates that there is no consistent calibration error in the height of the GNSS measurements from the robot. For this reason, no uniform offset was applied to subsequent GNSS results in this research. However, the elevation measurements shown here serve as an example of the magnitude of variation that can be expected in later results.

The ME values are best explained by environmental factors, such as the R10's height approximate 2.0 m height giving it a clearer view of the sky and horizon than the robot's Zephyr 2 antenna, which has a height of approximately 0.726 m above the ground. This means that signals from satellites low on the horizon are more likely to be attenuated by surrounding trees. The R7 is an older receiver than the R10, so it's software may be less effective at compensating for errors introduced by the ionosphere or troposphere.

Horizontal GNSS accuracy was not tested, as it was not possible to horizontally position the robot on the control point with an accuracy that would not likely exceed the 8 mm horizontal error of the R7. In addition, the purpose of this calibration was primarily to check the vertical accuracy of the system as there are many more sources of vertical error than there are horizontal (see Chapter 3).

5.4 Summary of Robotic Mapping Prototype

To summarize, an autonomous surveying robot was developed from a "Jackal" UGV chassis. The chassis uses inbuilt wheel encoders, an IMU and a high-accuracy GNSS receiver to collect information about the world and it's position within it. The robot runs a version of the Robotic Operating System and is controlled remotely by a laptop connected over a WiFi network. The robot has been developed to have the following functionality:

- The UGV can localize itself in an outdoor environment by fusing IMU, encoder and GNSS data in an Unscented Kalman Filter, implemented by the `robot_localization` package.
- The UGV can autonomously navigate from one point to another in an obstacle-free environment, controlled with costmap and path-planning nodes implemented by the ROS navigation stack, specifically the `move_base` package.
- The UGV can autonomously navigate between multiple move goals in order, given a list of desired coordinates. This is achieved using the `actionlib` package as well as custom software (`waypoint_navigation`) written to manage sequential goals and convert conventional coordinates into the internal `map` frame.
- The UGV can measure and record the elevation of the ground it passes over using a virtual reference frame positioned between where all four wheels contact the ground.
- The Velodyne HDL-32 allows the UGV to collect point cloud scans of its surrounding environment at a rate of 10 Hz.
- With all of the above features, the UGV can be commanded to autonomously travel across an area, constantly measuring and recording it's position in UTM coordinates. This enables it to autonomously conduct a topographic survey.

The first version of this prototype (see Figure 5.2) was used in a basic comparison with conventional survey methods, the results of which are shown in Chapter 6. The second iteration of the prototype (with LiDAR and upgrade IMU) was used in all subsequent chapters.

6 | Topographic Surveying with Robot Localization

Answering the Research Question requires directly comparing the prototype mapping robot to conventional survey methods. This chapter contains the methodology and results of that experiment, using the first prototype version. Many of the decisions described in this chapter have been informed by the research provided in Chapters 2 and 3.

As already stated, the version of the robot used to conduct the experiment described in this Chapter is the first version of the robot, as described in Section 5.1.3 and illustrated in Figure 5.2.

6.1 Experimental Setup

6.1.1 Selection of Survey Methods

Several methods were chosen which best represent conventional survey methods. They are outlined in Table 6.1. These are the survey methods whose performance was compared to that of the UGV.

TABLE 6.1: Chosen conventional survey methods

Abbrev	Instrument	Operating Mode	Measurement Technique	Sampling Strategy
TS-Stop	Total Station	Prism	Stop-and-go	Grid
TS-Cont	Total Station	Prism	Cont. topo	Grid
TS-Scan	Total Station	DR scan	N/A	Grid
GNSS-Stop	GNSS	RTK	Stop-and-go	Grid
GNSS-Cont	GNSS	RTK	Cont. topo	Grid
UAV-P	Aerial System	Photogrammetry	N/A	Grid

Each method followed the same grid-based sampling strategy to keep their data density and distribution consistent. The reasoning for this decision has already been explained in Sections 2.7 and 3.6. Every method also used the UTM grid projection, based on

the WGS84 reference frame with a DMA10 geoid model. This was done to ensure compatibility with the ROS localization packages which use the UTM grid.

For the total station methods, a Trimble VX Spatial Station was used, with a standard survey rod and R10 360 Prism. For GNSS system methods, a Trimble R10 GNSS receiver was mounted on a survey rod as the GNSS "rover" and a Trimble Net R9 GNSS receiver was used as the "base". For the aerial photogrammetry method, a Trimble ZX5 Multirotor UAV with a Sony A6000 camera was used.

6.1.2 Test Environment Selection and Preparation

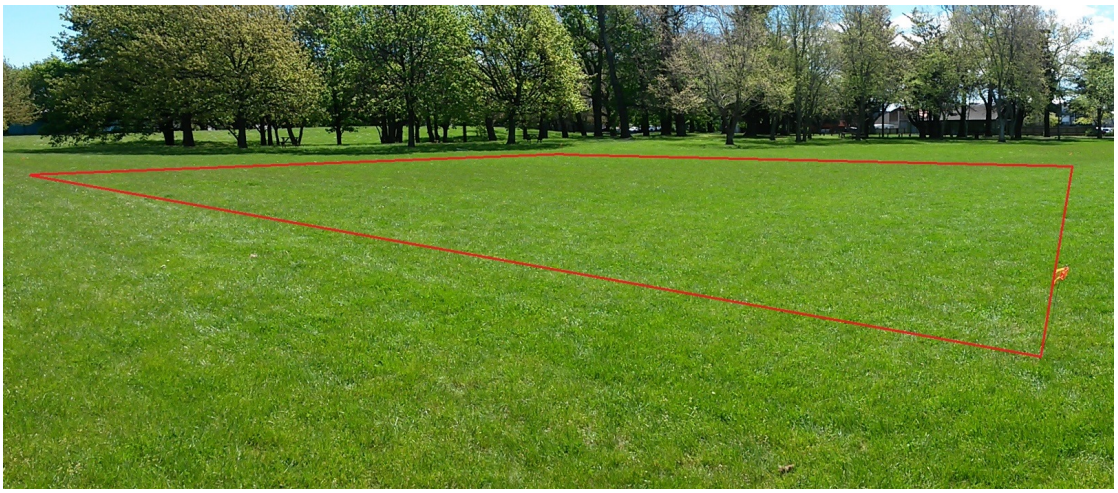
Several areas were considered as potential survey zones. They had to be in an easily accessible area that could be re-surveyed multiple times with the UGV and conventional survey instruments. This limited the test ground to parks and fields near the University of Canterbury campus or the Trimble Navigation premises. After considering the options available, two areas were chosen in Maryland's Reserve.

The first test ground was a 26x26 m square in a flat, open part of the reserve. This area, hereafter referred to as "Zone 1", represents an ideal survey environment with uniform topography and none of the features such as trees, vegetation or buildings that might be a source of error in the survey. The boundary of Zone 1 is shown in Figure 6.1(a). The second test ground was a 24x8 m rectangle on the side of a hill and bordered by trees. This area, hereafter referred to as "Zone 2" represents a more challenging survey environment with varied topography and environmental disturbance from nearby trees. The boundary of Zone 2 is shown in Figure 6.1(b)

These test grounds were selected to highlight the impact of slope and trees on survey instruments. To prepare these areas for surveying, each area was marked with control points. To do this four steps were followed:

1. The corners of each survey zone were roughly marked out with wooden pegs buried in the ground to act as boundary markers.

2. A control point was chosen for each survey zone and marked with a buried wooden peg. The total station used for all total station based survey methods would be set up above these points.



(a) Zone 1: Ideal survey environment

(b) Zone 2: Typical survey environment

FIGURE 6.1: Survey zones in Marylands Reserve

3. A coordinate was chosen and marked with a buried peg to act as the backsight control point for the total station. The same point was chosen to act as a backsight for both Zone 1 and 2.

4. A Trimble R10 GNSS survey system was used to survey each control point twice, at three different times of day to account for changes in satellite constellation (discussed in Section 3.4) . Each measurement had an observation time of three minutes. The results were then averaged to obtain accurate coordinates for each point.

Figure 6.2 shows an aerial view of each survey zone and the associated control and backsight points for each one.

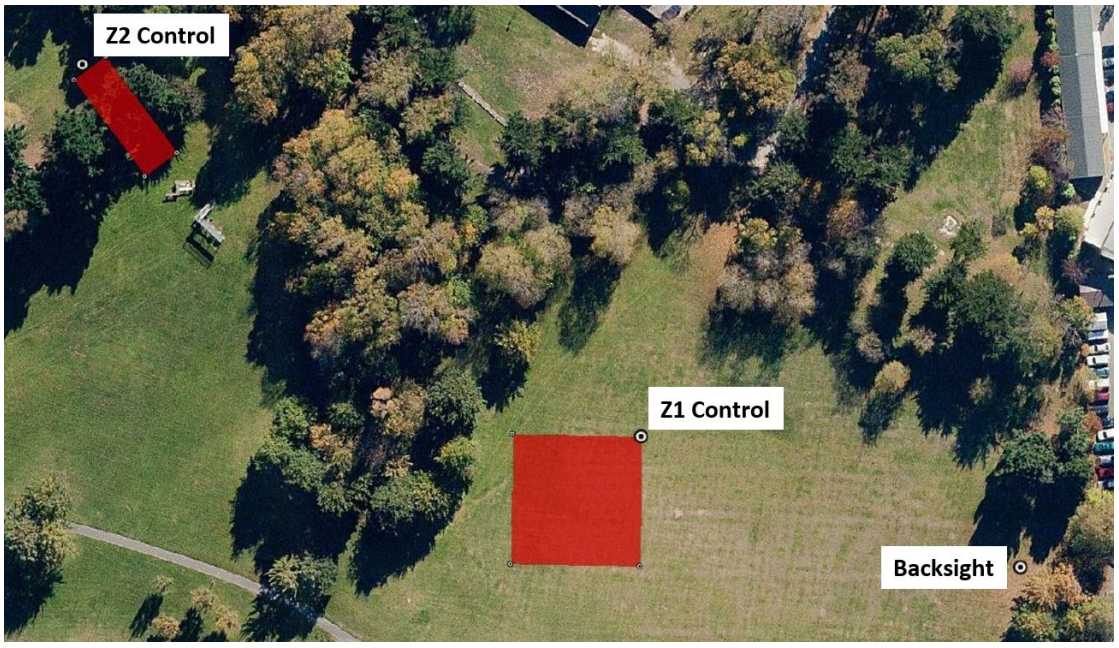


FIGURE 6.2: Aerial view of survey zones with control and back sites

6.1.3 Source of Ground Truth

Virtually every metric used to quantify accuracy is given as an error, i.e. the difference between a measured value and the true value. In the surveying context this is the difference between the measured topography and the true topography of an environment. However this requires a "ground truth" estimate of the topography that is as accurate as possible and is assumed to be error free [39]. Every measurement that deviates from this is in error *relative* to the ground truth.

Total stations have the highest point precision of any survey instrument and are widely used in academic research for establishing reference surfaces and ground truth estimates. Therefore, a Trimble VX total station was used to establish a ground truth for this research. To achieve this, a regular grid was measured out in each survey zone, and the vertices marked with spray paint. A 2x2 m grid of 198 points was used in Zone 1, and a 1x1 m grid of 225 points in Zone 2. Each marked vertex was surveyed three times across different days with a TS and prism using the stop-and-go measurement technique. This produced three separate measurements for each point in the grid, which were then averaged to create a mean estimate for the coordinates of each point. This collection of

points formed the ground truth datset for zones 1 and 2. The DEMs created from these data sets are provided in Figures 6.3 - 6.4.

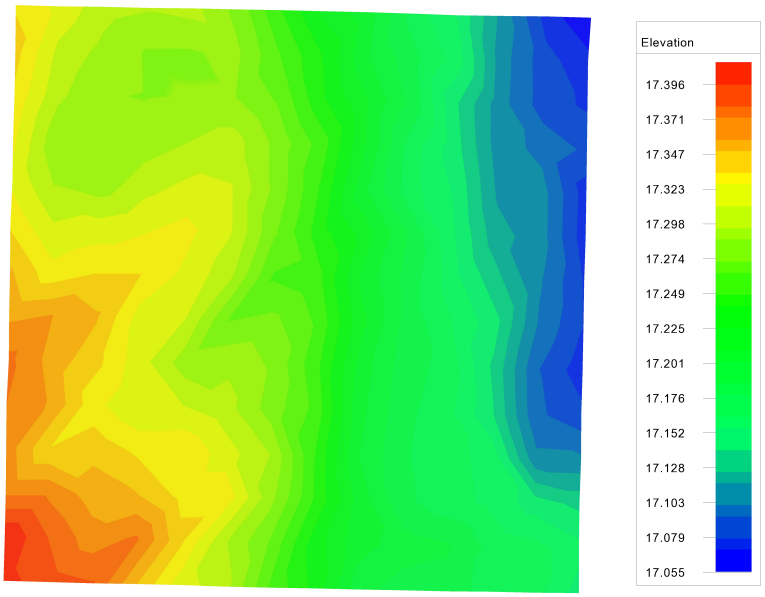


FIGURE 6.3: Ground truth DEM for Zone 1. Elevation is given in meters.

Note that the DEM for Zone 2 (Figure 6.4) uses the same colour range as Zone 1, but set to a different scale.

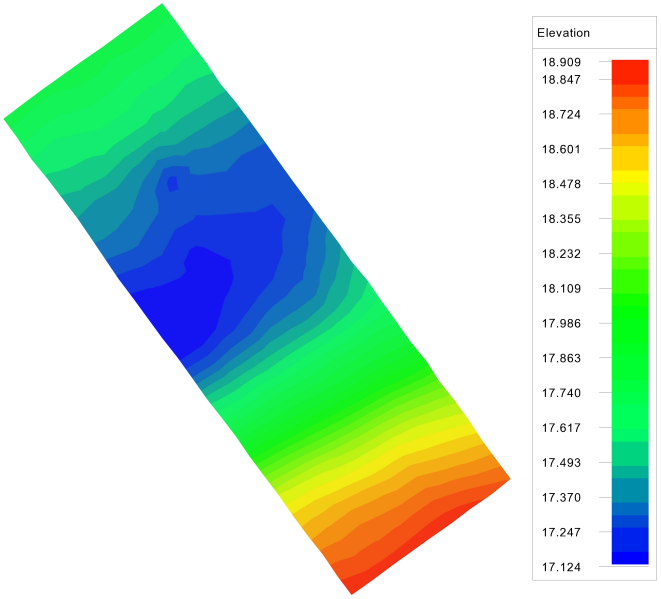


FIGURE 6.4: Ground truth DEM for Zone 2. Elevation is given in meters.

6.2 Proposed Methodology

6.2.1 Survey Data Collection

This section describes how each survey method was conducted. Each separate survey method was repeated three times for each survey zone. I.e. three TS-stop surveys for zone 1 and three for Zone 2, so every survey method produced six data sets in total. Unless stated otherwise, all surveys were conducted by the author. To keep data distribution and density consistent when surveying each site, the survey method in question always either surveyed the spray-painted grid vertices, or followed the grid lines.

Total Station Data Collection

A Trimble VX total station was used for all TS methods. In each case the station itself was set up on a heavy tripod over the control point for that zone. Regardless of the zone being surveyed, the same backsight was used to calibrate the station's azimuth. The rest of the survey procedure depended on the operating mode and measurement technique.

To take stop-and-go measurements a 2 m long survey rod was used with a Trimble 360 prism mounted on top. The TS was put in tracking mode and locked to the prism. Measurements were then taken at each grid-point previously established by the ground truth survey. I.e. measurements were taken at regular 1x1 grid intervals for Zone 1 and regular 2x2 grid intervals for Zone 2. The rod was carefully leveled with a calibrated bubble-level before taking each measurement. A flat foot was used on the survey rod rather than a spiked foot for all stop-and-go measurements. This is because in soft ground the tip can sink below the surface and give misleading results.

To take continuous topo measurements with the TS, the same setup procedure was followed, using the same survey rod and prism. But the operating mode was changed to "continuous topo" mode and set to automatically take measurements every 1 or 2 m depending on the zone. An extra 10 cm was added to the rod height, and the rod was held as consistently as possible at that height above the ground. The rod was then walked across the survey zone in a switch-back fashion, following the grid lines to complete the survey.

To take a DR scan, the TS was put in DR mode and the scan settings were selected so that the boundaries and resolution were the same as the ground truth data set (2x2 grid for Zone 1, 1x1 grid for Zone 2).

GNSS Survey System Data Collection

The GNSS system used in this research was always operated in RTK mode as a differential rover/base pair. The "rover" consisted of a Trimble R10 GNSS receiver (with integrated antenna) mounted on a 2 m long survey rod. The rover was used to carry out measurements while communicating with the base via radio. But instead of using a base set up on-site, a permanent base mounted on the roof of Trimble's Christchurch offices was used. This "base" consisted of a Trimble NetR9 GNSS receiver, Zephyr Geodetic 2 antenna, and a Trimmark 3 radio transmitter. As the offices are located adjacent to Marylands Reserve this proved to be a convenient alternative to a temporary base, and greatly simplified the data collection process. This base station was also close enough to the survey sites that the ± parts per million (ppm) component of the GNSS RTK error could be considered negligible.

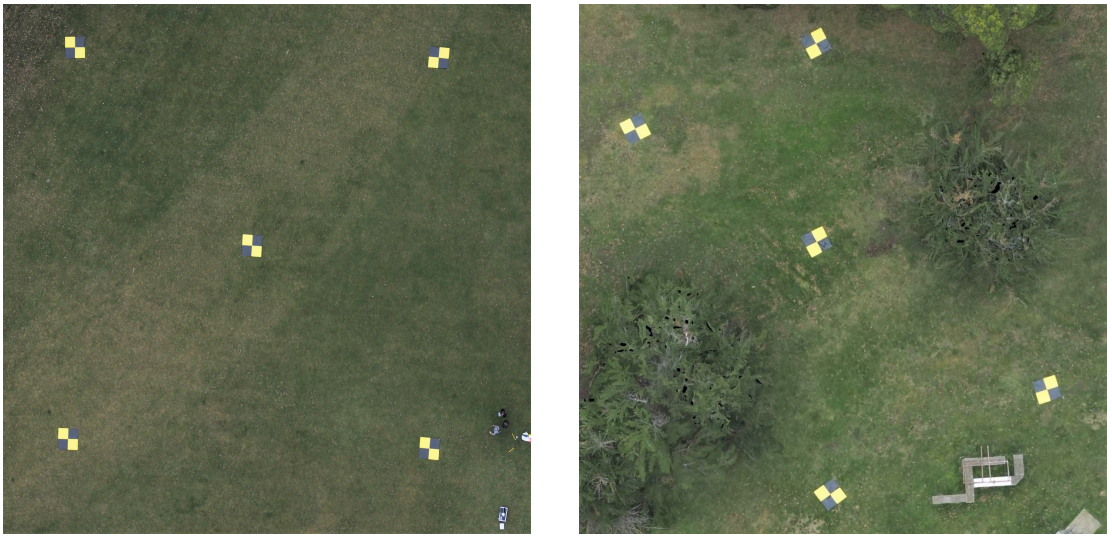
To take stop-and go measurements, the GNSS receiver was mounted on a 2 m long survey rod and configured to take measurements on command. Measurements were then taken at each grid-point previously established by the ground truth survey. The rod was carefully leveled before taking each measurement, and the observation time was set long enough to obtain three epochs for each measurement. As with the TS-stop method, a flat foot was used on the survey rod rather than a spiked foot.

To take continuous topo measurements, the GNSS receiver was set up on a 2 m long survey rod and configured to take measurements every 1 or 2 m depending on the zone. An extra 10 cm was added to the rod height, and the rod was carefully held at that height above the ground. The rod was then walked across the survey zone in a switch-back fashion, following the grid lines to complete the survey.

UAV Data Collection

The UAV-AP survey was carried out by a professional UAV pilot, using a Trimble ZX5 UAV with an aerial photogrammetry unit configured to take vertical photographs. Before beginning the AP survey for each zone, four GCPs were placed in the corners of the zone and one was placed in the middle, these are visible in the aerial photos provided in

Figure 6.5. A flight path was then programmed into the UAV which would take it across each survey zone. The UAV then autonomously completed the survey and the resulting images were processed in Trimble Business Center (TBC) to produce a point-cloud scan for each zone. Only one survey of each zone was completed using the UAV as it was not available to the project long enough to complete additional surveys.



(a) Zone 1 (b) Zone 2

FIGURE 6.5: Zone 1 and 2 photogrammetry images from the UAV-P survey method

UGV Data Collection

The UGV was configured to take continuous measurements, as having the UGV take stop-and-go measurements would offer no functional difference from a conventional GNSS stop-and-go survey. Whereas taking continuous measurements would maximize the advantages of the UGV platform: namely the fixed relationship of the GNSS antenna to the ground, and a dynamic position estimate from sensor fusion rather than a single sensor.

To actually conduct a survey, the UGV would start in one corner of the survey and autonomously travel back and forth in a switch-back fashion, following the grid lines and producing the trajectory shown in Figure 6.6. After completing the survey, the position data was filtered to make the data adhere to the same 1x1 or 2x2 grid spacing as the other survey methods.

All data was recorded in a ROS bag file, and later played back through an Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF), as implemented by the `robot_localization` package. How these filters were configured is described in more

detail in Section B.1.2. The parameters of the EKF and UKF were not tuned specifically for this research, and were instead left to their default values. This decision was made because EKF parameters are difficult to tune well, and have to be tuned by hand. In addition, the the `robot_localization` author advises against tuning some parameters unless the user is familiar with them [136].

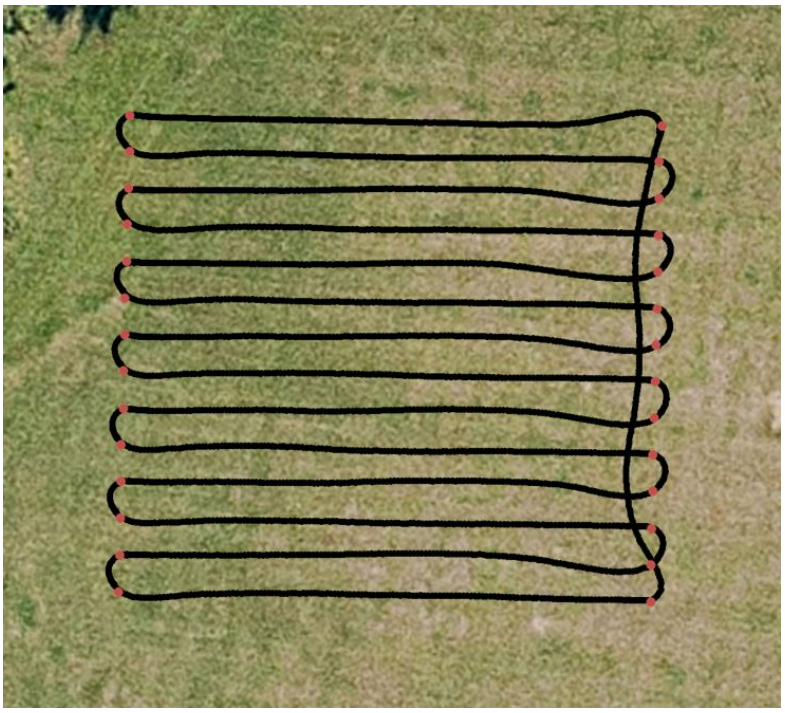


FIGURE 6.6: Example trajectory from autonomous navigation across an outdoor area. Red dots mark the vertices at the end of each grid row.

Regarding the setup of sensors on the UGV, the IMU and wheel encoders are independent, and do not require any additional setup other than calibration. However, the GNSS system mounted on the UGV was operated in RTK mode and therefore needed a GNSS base to communicate with. For this, the base unit mounted on the roof of the nearby Trimble offices was used, as per the other GNSS methods.

6.2.2 Data Processing

The data collected using each survey method is only raw coordinate information, i.e. a list of Northings, Eastings and Elevations. It had to be processed, interpolated and measured before it could be compared. To do this, three steps were followed:

1. **Create DEM:** Each set of raw data was imported into Trimble Business Center (TBC) as raw coordinates. TIN interpolation was then used to create a unique DEM for that data set. This produced six DEMs in total for each survey method, three for each survey zone.

2. **Create DoD:** Each DEM was subtracted from the ground truth DEM to produce a DoD for that data set. The a DEM of Difference (DoD) visually represents the difference between that data set and the best estimate of the true topography. This produced six DoDs in total for each survey method, three for each survey zone.

3. **Calculate Results:** The performance metrics were calculated for each data set, using the DEM, the ground truth data, and the equations described in Section 3.1.1. Specifically the equations for the Mean Error (ME), Root-Mean-Square Error (RMSE) and Standard Deviation (SD). The time taken to conduct each survey was also measured. This produce eight sets of metrics for each survey method, four for each survey zone. The results for each zone were then averaged to find the *overall* performance for the survey method in that zone.

6.3 Results

The statistical results of each survey method are provided in Table 6.2 for Zone 1, and Table 6.3 for Zone 2. The DoDs are also provided with one for each survey method.

6.3.1 Elevation Error Statistics

The raw results for each data set are shown to highlight the variation within different data sets for each method. Even though only three data sets exist for most methods, it gives an indication of the repeatability of each method, which is discussed further in the discussion (Section 6.4).

6.3.2 Difference Models

This section contains example difference models or DEMs-of-Difference (DoD) between each survey method and the ground truth. Figure 6.7 contains example DoDs for each

TABLE 6.2: Raw survey results for Zone 1

Method	Dataset	ME (mm)	RMSE (mm)	SD (mm)	Time[1] (min)
TS-Stop	1	2	2	1	70
	2	0	1	1	65
	3	-1	1	1	71
TS-Cont	1	46	51	23	4
	2	40	44	17	4
	3	24	33	17	4
TS-Scan (w/ grass)[2]	1	71	73	15	3
	2	70	71	16	2
	3	69	71	15	3
TS-Scan (no grass)[3]	1	1	15	15	3
	2	-1	16	16	2
	3	-1	15	15	3
GNSS-Stop	1	6	9	7	60
	2	5	9	7	47
	3	6	8	6	52
GNSS-Cont	1	56	64	31	4
	2	38	47	28	4
	3	69	75	29	5
UAV-P	1	2	14	14	1
Robot (GNSS)	1	5	12	11	13
	2	7	12	10	13
	3	8	14	11	13
Robot (EKF)	1	11	21	18	13
	2	8	16	13	13
	3	17	38	34	13
Robot (UKF)	1	6	12	10	13
	2	7	12	10	13
	3	11	19	16	13

[1] Does not include setup, flight-to-site or pack-up time as applicable
[2] Raw, as-measured data without compensation for grass height
[3] 70 mm offset subtracted from all elevation measurements to account for grass height

method in Zone 1, and Figure 6.8 shows similar examples for Zone 2. Although a DoD was created for every data set, the examples shown have been hand-picked because they are typical examples of the method or because they exhibit quirks of the specific method that was used to collect that data, which is then discussed later in Section 6.4. So unless otherwise stated, they have not been cleaned to remove anomalies or outliers, to better demonstrate the various sources of error for each method. Note that blue regions are

TABLE 6.3: Raw survey results for Zone 2

Method	Dataset	ME (mm)	RMSE (mm)	SD (mm)	Time[1] (min)
TS-Stop	1	4	4	1	68
	2	-2	2	1	69
	3	-2	2	1	56
TS-Cont	1	15	26	22	4
	2	8	21	19	5
	3	6	19	18	4
TS-Scan (w/ grass)[2]	1	66	70	22	3
	2	68	72	23	3
	3	67	71	23	4
TS-Scan (no grass)[3]	1	2	22	22	3
	2	3	23	23	3
	3	2	23	22	4
GNSS-Stop	1	15	22	17	100
	2	11	19	15	62
	3	19	32	26	74
GNSS-Cont	1	21	43	37	4
	2	42	55	36	4
	3	31	41	27	4
UAV-P	1	3	19	19	3
Robot (GNSS)	1	20	31	23	14
	2	20	22	19	14
	3	20	29	21	14
Robot (EKF)	1	23	38	30	14
	2	9	40	39	14
	3	20	37	31	14
Robot (UKF)	1	21	30	22	14
	2	8	28	26	14
	3	21	33	26	14

[1] Does not include setup, flight-to-site or pack-up time as applicable
[2] Raw, as-measured data without compensation for grass height
[3] 70 mm offset subtracted from all elevation measurements to account for grass height

where the method measured an elevation higher than the ground truth, red is where it measured a lower elevation, and the brighter the colour, the greater the difference.

Table 6.4: Mean survey results for Zone 1

Method	ME (mm)	RMSE (mm)	SD (mm)	Time[1] (min)
TS-Stop	0	2	1	69
TS-Cont	37	42	20	4
TS-Scan (grass)[2]	70	72	15	3
TS-Scan (no grass)[3]	0	15	15	3
GNSS-Stop	6	9	6	53
GNSS-Cont	54	62	29	4
UAV-P	1	2	14	1
Robot (GNSS)	7	13	11	13
Robot (EKF)	12	25	22	13
Robot (UKF)	8	14	12	13

[1] Does not include setup, flight-to-site or pack-up time as applicable
[2] Raw, as-measured data without compensation for grass height
[3] 70 mm offset subtracted from all elevation measurements to account for grass height

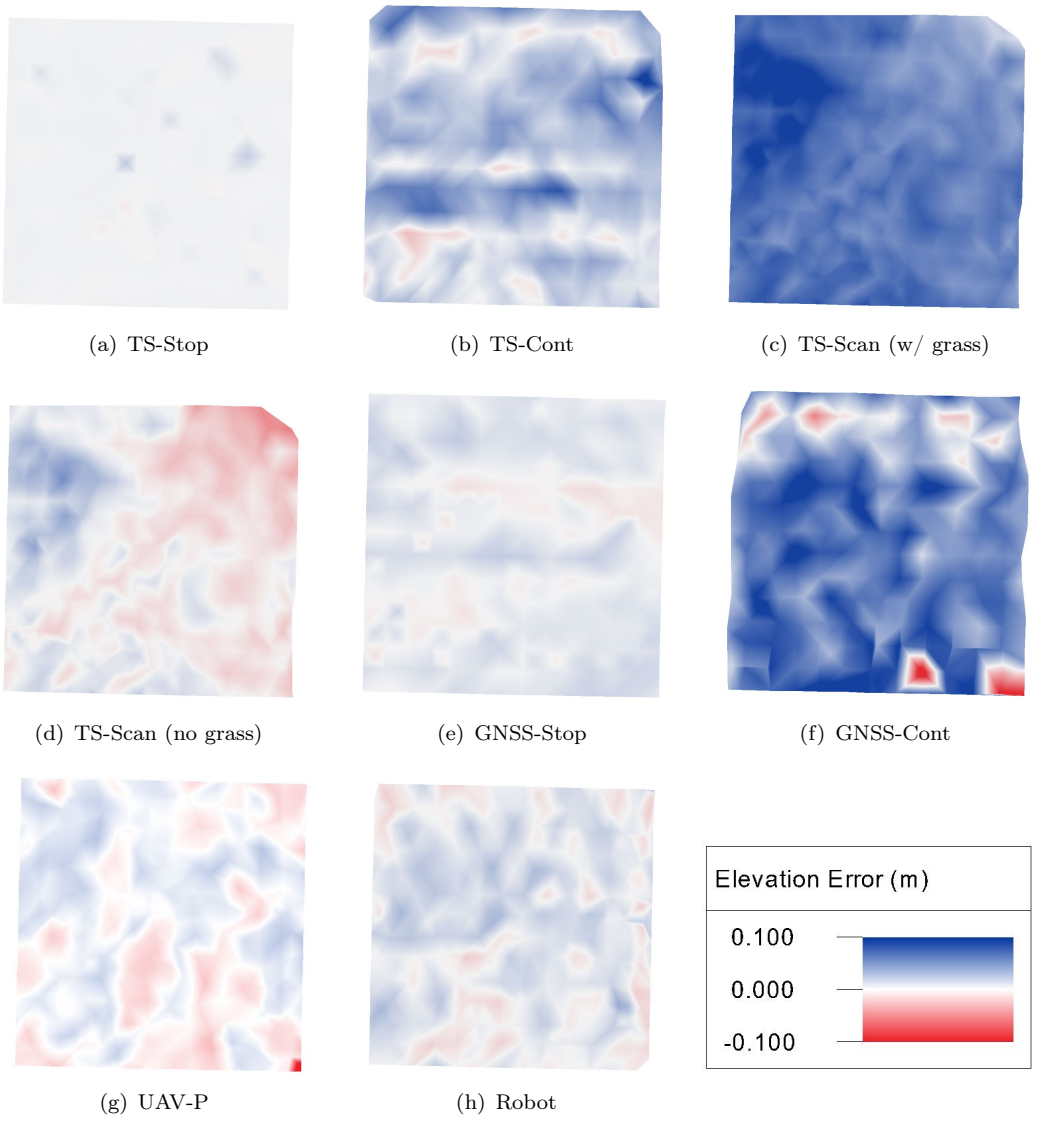
Table 6.5: Mean survey results for Zone 2

Method	ME (mm)	RMSE (mm)	SD (mm)	Time[1] (min)
TS-Stop	0	3	1	64
TS-Cont	10	22	20	4
TS-Scan (grass)[2]	67	71	22	3
TS-Scan (no grass)[3]	2	22	22	3
GNSS-Stop	15	24	19	79
GNSS-Cont	31	46	33	4
UAV-P	3	19	19	3
Robot (GNSS)	17	27	21	14
Robot (EKF)	17	38	33	14
Robot (UKF)	17	30	25	14

[1] Does not include setup, flight-to-site or pack-up time as applicable
[2] Raw, as-measured data without compensation for grass height
[3] 70 mm offset subtracted from all elevation measurements to account for grass height

6.4 Discussion

The variety in the results returned by each survey method reflect the instrument, their use, and several environmental effects. To properly compare each method, these factors must be disentangled and addressed individually.



(a) TS-Stop (b) TS-Cont (c) TS-Scan (w/ grass)

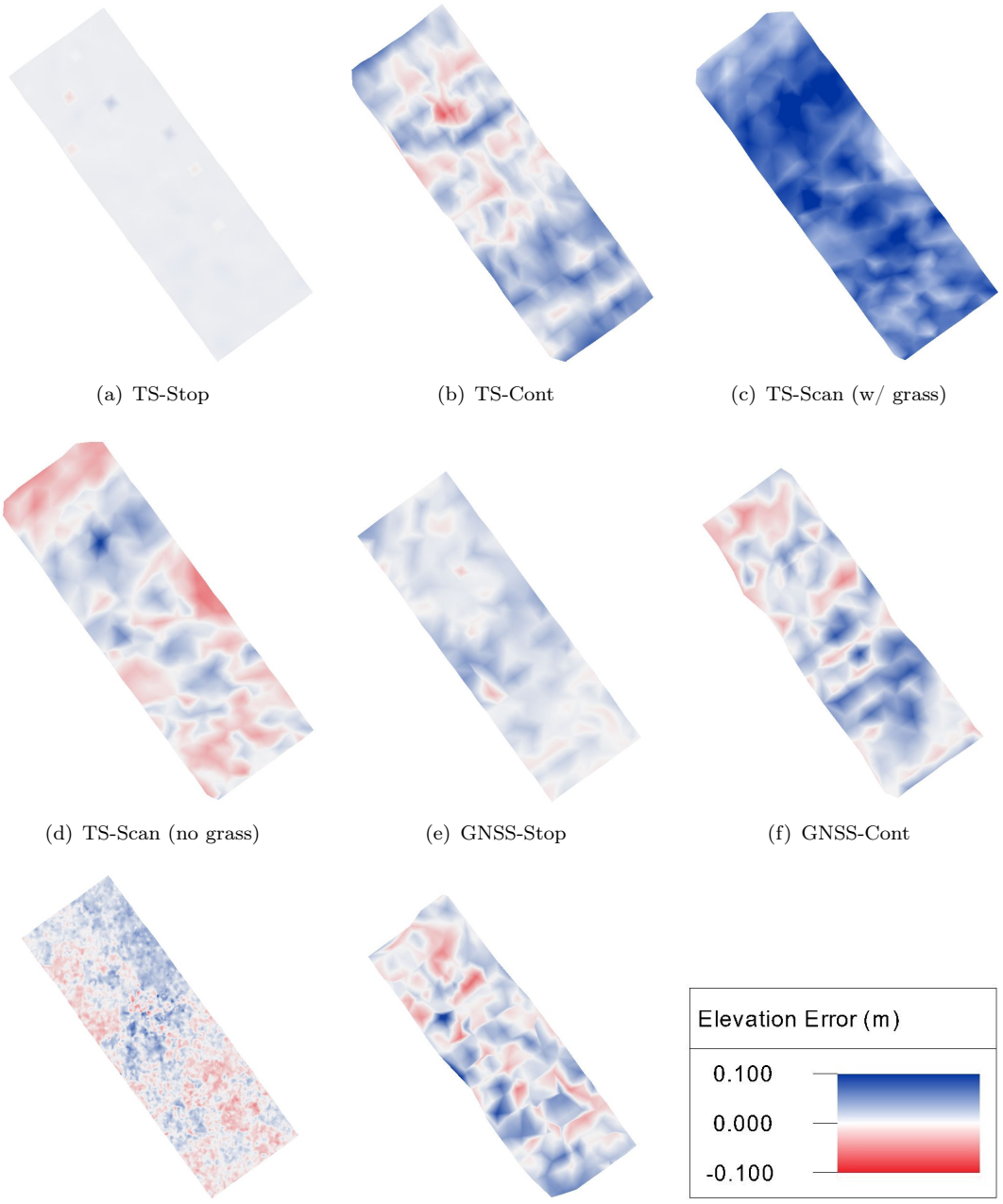
(d) TS-Scan (no grass) (e) GNSS-Stop (f) GNSS-Cont

(g) UAV-P (h) Robot

FIGURE 6.7: Zone 1 difference models

6.4.1 Effect of Measuring Technique

The two distinguishable measurement techniques are stop-and-go (TS-Stop, GNSS-Stop) and continuous topo (TS-Cont, GNSS-Cont, all Robot results). Their results largely follow the intuition that a method which precisely positions the measuring instrument will be more accurate than one which does not. The TS-Stop and GNSS-Stop results are more accurate than their continous topo equivalents in both zones, as shown in Tables 6.4 - 6.5. For the same reason, the stop-and-go datasets are more closely grouped than the TS/GNSS-Cont and Robot results. Each set of results are within 1-2 mm RMSE of each other in both zones (Tables 6.2 - 6.3).



(a) TS-Stop (b) TS-Cont (c) TS-Scan (w/ grass)

(d) TS-Scan (no grass) (e) GNSS-Stop (f) GNSS-Cont

(g) UAV-P (h) Robot

FIGURE 6.8: Zone 2 difference models

When using the continuous topo technique it is virtually impossible for the surveyor to hold the rod perfectly vertical and at a consistent height above the ground. So the movement of the rod adds a significant amount of random error to the survey results that is avoided in the stop-and-go method. This error is visually represented in Figures 6.7(b) and 6.7(f), where lines of consistent colour can be seen. These lines show where the survey rod was consistently held too high or too low for a particular pass across the survey area. The difference in ME, RMSE and SD between the stop-and-go/continuous

topo techniques show how the surveyor can be a larger source of error than the instrument itself.

The clear trade off is that the stop-and-go methods take significantly longer to complete, typically taking approximately 60 minutes vs the average of 4-13 minutes for the continuous topo and Robot methods. This simply reflects the nature of each measurement technique.

Although the GNSS-Cont and Robot (GNSS) methods use different GNSS rovers (Trimble R10 vs Trimble R7 respectively), they used the same base station, and were both set to RTK mode. So their GNSS performance can be considered to be equivalent. They use the same continuous topo measurement technique and yet largely the Robot bypasses the draw-backs of the method because its GNSS antenna is always a fixed distance from the topography. As a result the Robot is more accurate with 13 mm vs 62 mm RMSE in Zone 1 and 27 mm vs 46 mm RMSE in Zone 2 (Tables 6.4 - 6.5). This shows that for a continuous topo measuring technique, a mobile robot will be more accurate than a human surveyor. However, the Robot (GNSS) results are still not as accurate and precise as the stop-and-go methods.

It must also be noted that methods which use a continous topo measuring technique - TS-Cont and GNSS-Cont are both more accurate in Zone 2 than Zone 1 (42 to 22 mm RMSE and 62 to 46 mm RMSE, respectively). This is because Zone 2 was surveyed chronologically after Zone 1, by which point, I had had more practice at holding the survey rod steady. This highlights how much the method depends on a surveyors skill, and how significant human error can be in the survey process.

6.4.2 Effect of the Environment

There are two environmental factors at play in the test sites. First, the presence of grass which affects remote scanning methods. Second, the presence of trees in Zone 2 which affects all GNSS-based methods.

As explained in Chapter 3, the presence of vegetation such as grass or bushes has a significant impact on the accuracy of the TS-Scan data. Even relatively short grass can have a noticeable impact as the uncorrected data has a 71-72 mm RMSE across both survey zones. This is shown by the consistent dark blue colour in Figures 6.7(c) and

6.8(c), indicating that the measured elevation was consistently much higher than ground truth. The average offset across each survey zone was calculated by measuring the grass height in various locations across each survey zone and averaging it. Once the vertical offset has been removed from the measured data, the TS-Scan data becomes far more accurate - around 15-22 mm RMSE. This is a well-known fault with any remote-scanning survey instrument and, as in this case, requires the surveyor to mark and measure various terrain features to be removed in post-processing.

Trees have clearly affected the GNSS methods in Zone 2. The trees block line of sight between the GNSS rover and satellites, predominantly blocking satellites that were lower in the sky and would have otherwise provided a broader satellite geometry (similar to Figure 3.1(c)). In zone 1 approximately 13-15 satellites were in view and the VDOP was generally <1 (rated as "Ideal"). In zone 2 10-12 satellites were in view, but because of the trees they occupy a narrower region of the sky. This increases the VDOP in some parts of zone 2 to approximately 3.0. Although a DOP value of 3.0 is still rated as "Good" according to Table 3.2, the GNSS measurements are not as accurate overall. As a result, GNSS Stop/Cont and all Robot methods show higher RMSE and SD errors in Zone 2 than in Zone 1.

6.4.3 Extended vs Unscented Kalman Filter

As discussed in Section 4.3.1, the UKF is generally considered to be superior to the EKF, and this is reflected in the data. The UKF results are more accurate than the EKF in both test sites, with all metrics. This makes intuitive sense as the EKF is known for being poor at estimating non-linear motion. While both test sites may appear to be relatively flat, the ground is rough, and the robot lacks any suspension. So it vibrates and moves in a highly non-linear way when travelling at speed. This is made clear when inspecting a segment of the trajectory in Figure 6.9. In this Figure, the time segments where the robot was moving are coloured cyan. During these segments, the UKF follows the GNSS elevation more accurately.

It is also worth noting that the raw GNSS results initially appear to be slightly more accurate than either the EKF or UKF. In the EKFs case, this is because the algorithm does a poor job of following the GNSS odometry, as already shown in Figure 6.9. In the case of the UKF, this is due to slight outliers in the data, which can be observed

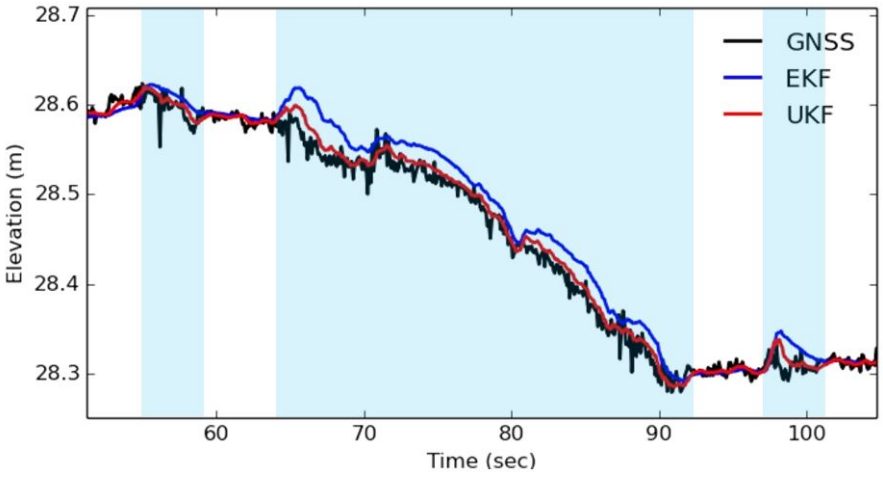


FIGURE 6.9: UKF vs EKF elevation estimation during robot motion

in Tables 6.2 - 6.3. In the results for zone 1 (Table 6.2), the 3rd UKF data set is a slight outlier with a higher ME, RMSE and SD than the previous two data sets. This increases the averaged result, whereas if this data set was to be discarded, the averages are approximately the same as the raw GNSS data. In zone 2 (Table 6.3) a similar event has occurred, except the outlier is a GNSS data set that pulls down the average error.

6.4.4 Conventional vs Robotic Survey Performance

The robot data provides an interesting comparison with the TS and GNSS continuous topo survey methods. It effectively replicates the same measurement technique, but is more accurate and consistent because the roll/pitch/yaw of the antenna is accounted for, and because it is held at a fixed distance from the ground (the Trimble E10 does not have tilt compensation). This reinforces the idea that the way the instrument is handled is just as, if not more, important as the instrument itself.

Another advantage of the robot is that like the stop-and-go methods, it is in constant contact with the ground so that it is not affected by the height of underlying vegetation (such as grass) which can affect the accuracy of remote scanning methods like the TS-Scan method.

In terms of speed, the robot strikes a balance between the different extremes presented by the conventional survey methods. It is significantly faster than the stop-and-go methods (60 min vs 14 min), while being somewhat slower than the other methods which average 3-4 min. The method with the best error-to-time ratio is still the UAV-P method but

the robot may still be preferable in some situations due to its ability to be operated fully autonomously.

6.5 Summary of Surveying Results with Robot Localization

The data in Tables 6.2 and 6.3 answer the research question *"How does a topographic survey conducted by an autonomous mobile robot compare to a survey conducted by conventional methods"*. The short answer is that in ideal conditions an autonomous robot can conduct a topographic survey with a level of accuracy and precision that is competitive with some of the most accurate survey methods that are commonly used in industry. It can also complete a survey faster than several conventional methods with the added benefit of full autonomy, which means that once set up, it can continue to survey with little oversight.

However the robot is not without its own disadvantages. The current prototype has limited reliability and is poorly optimized, making it highly dependant on accurate GNSS data (shown by the decreased performance in Table 6.3). Environments which reduce the accuracy of the GNSS receiver affect both the localization and navigational accuracy of the robot. In general the robot may also not be suitable in especially rough or obstacle-rich environments. The robot is still limited by the chassis, which will not be able to traverse some obstacles such as fences, rivers or boulders. Like all survey instruments, it will be up to the surveyor to determine when and where to use the robot.

The limitations of the chassis can be overcome to some extent by adding ranged depth sensors to the platform, and scanning the environment rather than just traversing it. The limitations of the initial robot prototype, and the potential of the platform with 3D LiDAR is what motivated the development of the second prototype described in Section 5.1.4, and the research conducted in the remainder of this thesis.

7 | Effect of Numerical Imprecision on Mapping

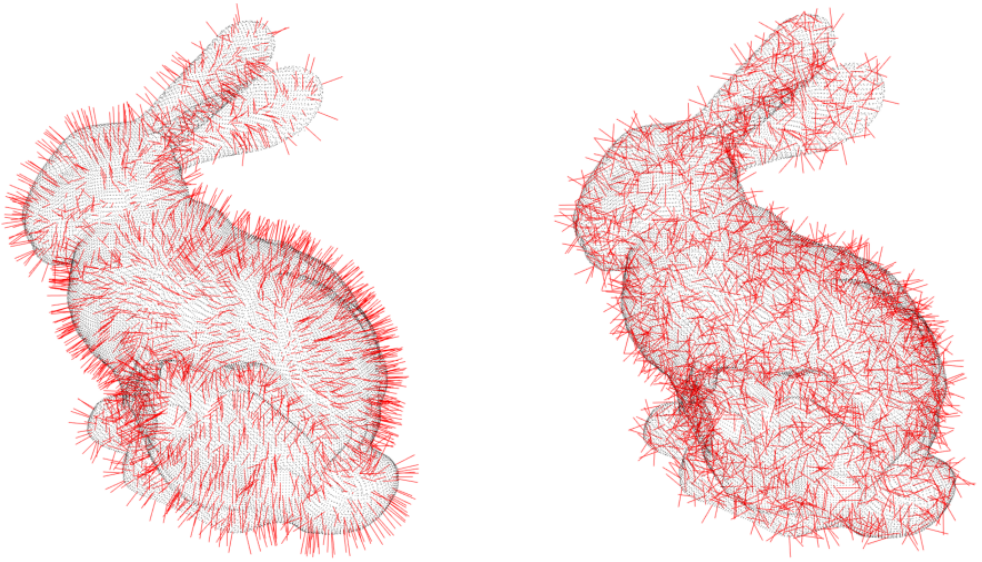
Much research has been done to date on the development of point cloud registration and normal calculation algorithms. Yet relatively little research has been done to ensure that the way they are implemented is robust and that their results are reliable.

While many algorithms for point cloud processing have infinite precision in theory, in practice they often require making compromises between precision and computation speed. This can introduce problems with loss of significance (LoS), where errors can be introduced that may have consequences for every down-stream process. This is of particular significance for normal calculation since this is often a precursor to other cloud processing operations such as feature identification or surface reconstruction.

Specifically, LoS occurs when the result of an arithmetic operation is too big to be accurately stored in the container type assigned to it. The result is then rounded or clipped to a number that the container can represent. Unfortunately the problem does not end there as this erroneous number may be involved in further arithmetic, propagating the error through an entire function and adding a considerable amount of error to the output.

In the context of cloud registration, this problem will manifest as a poor cloud match. In the context of normal vector calculation, the error may be obvious such as a "NaN" (Not-a-Number) or "Inf" (Infinite number, typically the result of a division by zero). Or the error may have an angular deviation from its true value.

Loss of Significance is easy to observe with a simple point cloud, such as the Stanford Bunny. When the cloud is at origin the normals (represented as red lines) all point outwards from the center of the cloud. But when the cloud is pushed an arbitrarily large distance from the origin, as shown in Figure 7.1(b), the normal vectors now appear to have random orientations.



(a) Correct normal orientation

(b) Incorrect normal orientation caused by loss of significance

FIGURE 7.1: Demonstration of how loss of significance occurs when a cloud is pushed far from the origin of its coordinate system, and the effect this has on point normal orientation.

7.1 Covariance Calculation in PCL

One of the most impactful places where LoS can occur is in the calculation of a covariance matrix. This is because they are highly sensitive to rounding error and because the covariance matrix of a cluster of points provides a measure of the spread, or randomness of their positions. So it is a fundamental basis of many other point cloud calculations, such as point normals, cloud registration [82] and cloud entropy [137].

The open-source Point Cloud Library (PCL) [138] is an extensive library for processing point cloud data. It is widely used in academia for a variety of computer vision, mapping and robotic applications. Issues of numerical precision and LoS are a concern for all point cloud processing software, and PCL is no exception. Because of its open-source and easily accessible nature, PCL is used by students and others who are relatively inexperienced with these issues, and so are often unaware of the potential pitfalls and care that needs to be taken in some point cloud applications. Particularly, applications involving LiDAR and large real-world data sets, where the magnitude of the point coordinates makes the clouds susceptible to LoS.

Further compounding the problem is the fact that it is not common practice in this field for users to visualize and explicitly check their point normal orientations to ensure they are accurate. As for registration, if a sub-optimal cloud match is obtained, it is often assumed to be the result of poor tuning or cloud quality, not a subtle failure on the part of the implemented algorithm. With this in mind, the contribution of this chapter is as follows: it explains how LoS can occur in normal vector calculation and cloud registration. It demonstrates this phenomenon with the open-source Point Cloud Library. It then proposes several best-practice recommendations for mitigating potential loss of significance.

It should be noted that while this chapter refers exclusively to PCL, and describes what should be considered an extreme case, LoS will affect all cloud processing software to a greater or lesser degree.

The covariance of two discrete, non-continuous variables (in this case any combination of p_{ix}, p_{iy} or p_{iz}) is defined as the mean product of the difference between each variable and its mean, as shown in Equation (7.1). This can be expanded out using the linearity property of expectations.

$$\begin{aligned} cov(x, y) &= E[(x - E[x])(y - E[y])] \\ &= E[xy] - E[x]E[y] \end{aligned} \tag{7.1}$$

A version of the expanded form of Equation (7.1), as implemented by PCL, is given in Algorithm 1. In practice, the functional code calculates the covariance of each combination of coordinates ($cov(x, x)$, $cov(x, y)$, $cov(x, z)$, etc.) and the covariance matrix itself, all within the same function. The specific section of PCL code that corresponds to Algorithm 1 is the `computeMeanAndCovarianceMatrix` function, defined in the source file `centroid.hpp`.

Algorithm 1 Covariance calculation.

Input: A cloud of N points

Output: Covariance of x and y for the cloud

1: initialize array $mean$
2: **for** $point$ **in** $cloud$ **do**
3: $\quad mean[0] = mean[0] + point.x$
4: $\quad mean[1] = mean[1] + point.y$
5: $\quad mean[2] = mean[2] + point.x * point.y$
6: **end for**
7: $mean = mean/size\_of(cloud)$
8: $cov = mean[2] - mean[0] * mean[1]$
9: **return** cov

7.1.1 Single precision and Loss of Significance

In mathematics, the decimal precision of any given value can be arbitrarily large. However, in software, a value must be stored in a variable of a finite size. For reasons of computational efficiency, most point cloud processing is done in the C or C++ languages with 32-bit, single-precision, floating point numbers as defined by the IEEE 754 standard [139]. That is, each point coordinate (p_{ix}, p_{iy} or p_{iz}) is stored as an individual "float" which can accurately represent seven or fewer significant digits [140]. Some functions in PCL and other point cloud processing libraries such as Libpointmatcher give the user the option to use 64-bit, double-precision numbers ("doubles") [141]. Although as this chapter will show, increasing the size of the variable data type simply increases the threshold at which LoS will occur.

Computer code such as that shown in Algorithm 1 is particularly susceptible to LoS not only because it is summing many (potentially large) point coordinates but because it is summing the result of their multiplication. A full numerical analysis of the relevant parts of the PCL code base is well beyond the scope of this research, but a small example with two points is useful in illustrating the effect. A reader interested in learning more about numerical precision should refer to the work by Goldberg [142].

Let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be two points whose x and y coordinates (in meters) require exactly seven significant digits to be accurately stored to the nearest millimeter:

$$p_{1x} = 6273.544 \quad p_{1y} = 5180.157$$
$$p_{2x} = 6273.794 \quad p_{2y} = 5180.657$$

If these points are input to Algorithm 1, we see that some arithmetic operations produce an incorrect result, as shown by Table 7.1 where the incorrect digits are underlined.

In this example, the errors that occur are on the order of centimeters. However, it is clear that the more points there are in the cloud or the further it is from origin, the greater the LoS will be. As shown below, the problem is also a function of point spacing. Because the smaller the difference is between point coordinates, the more severe the error is when significant digits are lost.

When Algorithm 1 is used in practice, the size of the input cloud is determined by the number of nearest neighbors k required to calculate the normal. Since this number is usually important to the accuracy of the calculated normal, it can be treated as a constant, in which case the LoS is *effectively* a function of just the point distance and spacing.

LoS also occurs at various stages of each ICP variant, as their implementations also require a significant number and variety of arithmetic operations. Their implementation in PCL is significantly more complex than that of point normal calculation. Thus, while the next section explains how LoS occurs during normal calculation in detail, a similar analysis of the ICP variants is beyond the scope of this thesis.

TABLE 7.1: Example of Loss of Significance in Algorithm 1 when calculated with Single Floating-point Precision

Line	**Operation**	**Correct Result**	**Single Precision**
3	$\sum p_{ix}$	12547.338	12547.33<u>7</u>
4	$\sum p_{iy}$	10360.814	10360.814
5	$\sum p_{ix}p_{iy}$	65000320.000	6500031<u>7.669</u>
7	$0.5\sum p_{ix}$	6273.669	6273.669
	$0.5\sum p_{iy}$	5180.407	5180.407
	$0.5\sum p_{ix}p_{iy}$	32500160.000	3250015<u>8.825</u>
8	See Equation (7.1)	−3612.209	−3612.<u>000</u>

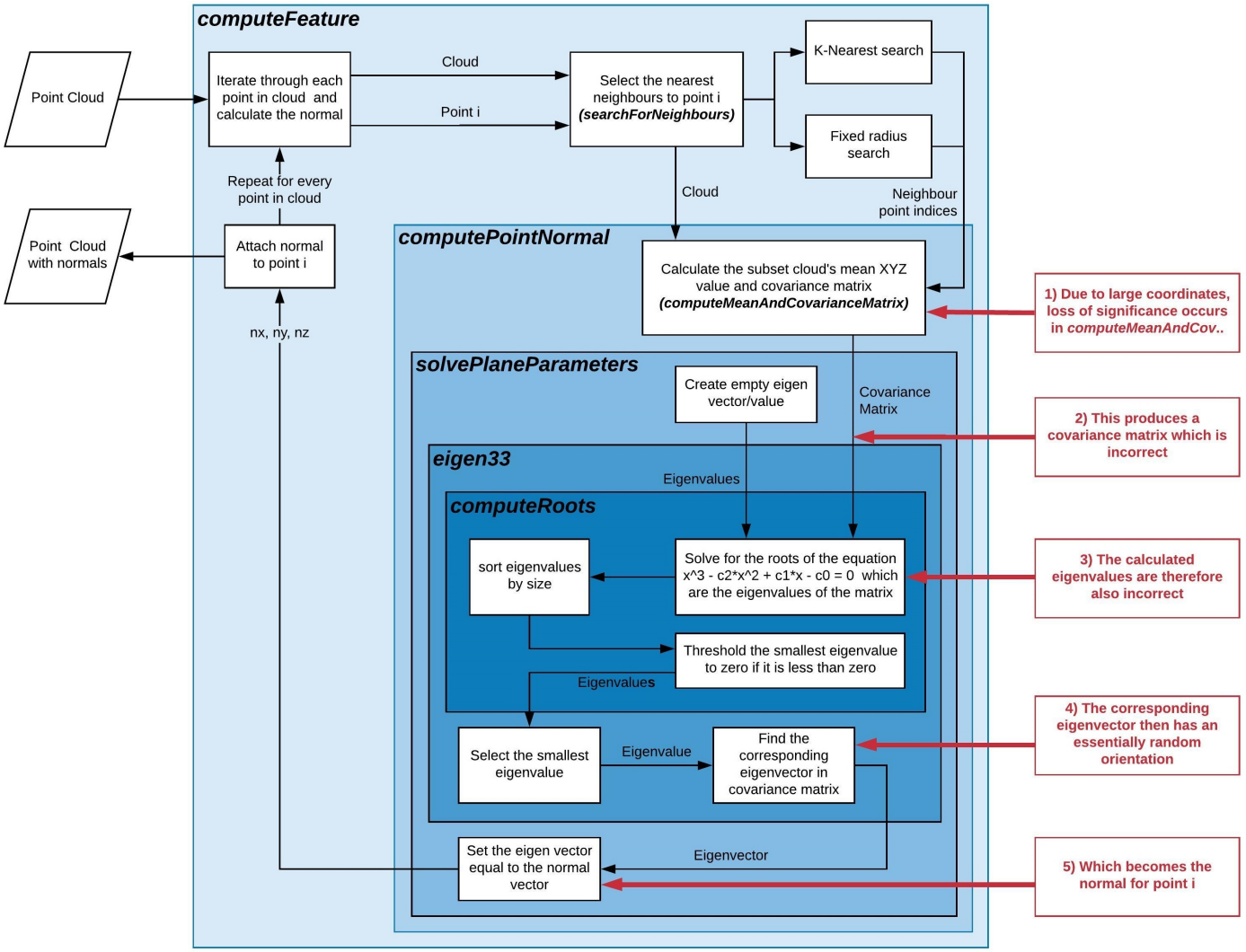


FIGURE 7.2: Flow diagram outlining the PCL process for calculating point normals and how LoS causes them to be calculated inaccurately. The key stages of the numerical precision issue are shown in the red text boxes. Names of the relevant functions are shown in bold italics.

7.1.2 Loss of significance in PCL calculation of normal vectors

The point normal calculation process in PCL is illustrated in Figure 7.2, as it is implemented at the time of writing. This process and individual function names may change after publication, however this figure has been included because it is invaluable in illustrating the problem.

When LoS occurs, it can change the value of the terms in Equation (4.2). This means that the result can be wrong. However, it also means that, when computing the variances along the diagonal of the covariance matrix $\mathbf{C}$, the $E[xx]$ (or $E[yy]$ or $E[zz]$) term can be rounded down and become smaller than the $E[x]E[x]$ term, resulting in a negative

variance which is theoretically impossible. The characteristic equation for $\mathbf{C}$ is sensitive to error in every element of the matrix. So when LoS occurs and an element is calculated incorrectly the resulting eigen values and vectors of $\mathbf{C}$ have a highly random nature.

As stated above, the problem is a function of cloud distance from origin and point spacing, and that it has a negative effect on cloud registration. Empirically demonstrating this effect is the focus of the remainder of this chapter.

7.2 Experimental Setup

7.2.1 Choice of test data sets

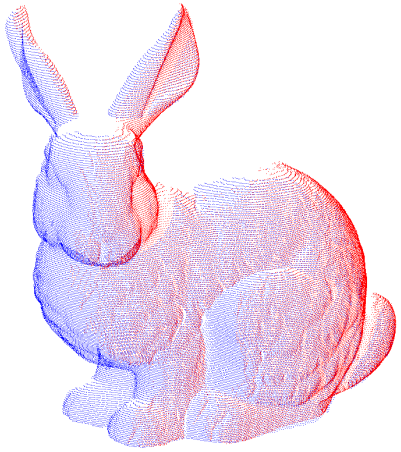
When choosing point cloud datasets with which to conduct empirical experiments, some researchers use publicly available data, their own generated clouds [71, 74], mathematically defined clouds [73] or a combination of the three [72, 78].

For this research, a selection of three fabricated and real-world point clouds were chosen that reflect some of the common applications PCL is used for. Namely, the well-known Stanford Bunny to reflect the common use of PCL in manipulating small object point clouds, a LiDAR scan from a Velodyne HDL-32 to represent PCL's use in SLAM, and a Trimble SX10 scan to reflect the uses of PCL in geoscience and mapping. These clouds are shown in Figure 7.3.

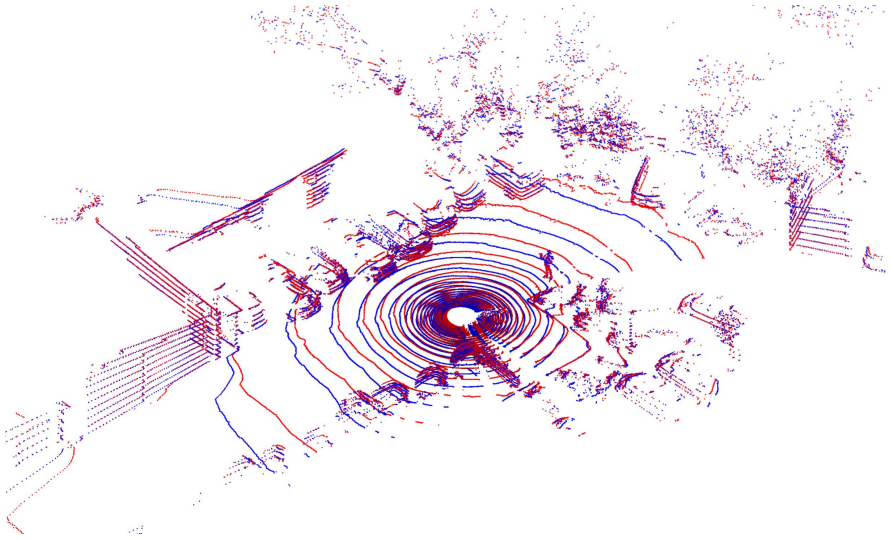
All three sets consist of an A and B cloud, which overlap but are non-identical. The Stanford Bunny dataset is composed of two of the original scans from the Stanford 3D Scanning Repository [143], specifically scans 45 and 315, which have a size of approximately 40,000 and 35,000 points, respectively. They have not been scaled in any way, and so retain their original dimensions of approximately 0.15 m $\times$ 0.15 m $\times$ 0.12 m.

The Velodyne HDL-32 carpark scans have dimensions of 43 m $\times$ 26 m $\times$ 5 m and a size of approximately 56,000 points each. The Trimble SX10 clock tower clouds have dimensions of 49 m $\times$ 40 m $\times$ 10 m and a combined size of approximately 4.5 million points.

Each pair of scans has already been correctly positioned relative to one another. Either by registering them with another application (Stanford Bunny), GPS positioning (HDL-32 scan) or total station positioning (SX10 scan).



(a) Dataset 1: Scans 45 and 315 of the Stanford Bunny



(b) Dataset 2: Two Velodyne HDL-32 LiDAR scans of a carpark

(c) Dataset 3: Two Trimble SX10 LiDAR scans of a clock tower

FIGURE 7.3: Test data sets. Each data set has been chosen to represent a type of point cloud and application that PCL is commonly used for. Each data set consists of an A (red) and B (blue) cloud which overlap but are non-identical.

It should be noted that these clouds are expressed in units of meters, as this is the native unit that the HDL-32 and SX10 datasets are created in. However, in the context of numerical precision, the units are arbitrary as the number of significant digits required to accurately store a number is the same regardless.

It is also worth noting that basic point clouds with mathematically definable normals (e.g. sphere, cube, and pyramid) were deliberately not selected. This is because the point normal calculation methods described and tested in this chapter would inherently produce slightly different normals, particularly near sharp edges of the object. Thus, any results would show some amount of non-zero error that was not induced by LoS.

7.2.2 Error Metrics

Normal Orientation Error: If the ground truth normal vector of a point is $\boldsymbol{n}_i$, then the angular error in radians between it and the calculated normal vector $\hat{\boldsymbol{n}}_i$ is given by Equation (7.2). Note that both vectors must be normalized.

$$\theta_i = arccos(\hat{\boldsymbol{n}}_i^T \boldsymbol{n}_i) \tag{7.2}$$

Mean Point Spacing: Mean point spacing $\overline{s}_i$ is defined as the mean Euclidean distance between any point $\boldsymbol{p}_i$ and its nearest k neighbors in the set Q_i, formally given by Equation (7.3).

$$\overline{s}_i = \frac{1}{k} \sum_{j=1}^{k} \|\boldsymbol{p}_i - \boldsymbol{q}_{ij}\|_2 \tag{7.3}$$

Each of these metrics can be calculated per-point, where i denotes the point index, or they can be averaged over the whole cloud to produce a per-cloud mean. Note that calculating the mean point spacing for a whole cloud is only valid if the points are approximately uniformly distributed over the surface of the object, i.e., in the case of the Stanford Bunny clouds.

Distance-Density Ratio: This chapter also introduces the concept of a distance–density ratio, which is simply the Euclidean distance of the point from origin, divided by the mean point spacing.

$$DDR_i = \frac{\|\boldsymbol{p}_i\|_2}{\overline{s}_i} \tag{7.4}$$

As shown Section 7.3.2 , the larger the ratio is the more likely it is that LoS will occur and the more severe it will be.

Registration Error: Each dataset consists of two overlapping, non-identical clouds that are already correctly positioned relative to one another. When testing a cloud registration algorithm, a copy of cloud B is made and translated a known distance from its original position, this is the initial error. The error-added copy, B′ is then registered to cloud A using one of the ICP variants described in Section 4.2.2. If the registration is perfectly accurate, registered cloud B′ should exactly match cloud B. If not, there will be some non-zero error that can be described as the mean Euclidean point-to-point distance between each point $\boldsymbol{p}_i$ in cloud B and the same point in the registered cloud B′:

$$\bar{\varepsilon}_r = \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{p}_{ib} - \boldsymbol{p}_{ib'}\|_2 \tag{7.5}$$

As with all other distances, registration error is expressed in meters.

7.2.3 Source of Ground Truth

The "true" orientation of any normal vector can only be known if it can be determined mathematically (as with a sphere, cube or plane). Virtually all normal calculation algorithms rely on the surrounding points to define the local curvature of the shape, and therefore rely on the point accuracy and density. Error, either added deliberately or acquired from sensor noise, makes determining the true normal impossible in most circumstances. Thus, what is calculated instead is simply the "best achievable" estimate of the "true" or "ground-truth" point normal. This is the case for all datasets shown in Figure 7.3.

In the context of *this* chapter, the ground truth for each point normal is still calculated using the PCL process described in Section 7.1.2 and illustrated in Figure 7.2. However, for each "true" normal $\boldsymbol{n}_i$ to be calculated, the relevant point $\boldsymbol{p}_i$ and its nearest neighbors Q_i are de-meaned by subtracting the coordinates of $\boldsymbol{p}_i$. This process centers each cluster of points on the origin and sets the DDR to zero, thereby minimizing any possible LoS. This is the sole method used for computing the ground-truth normals for every cloud in every test. To reiterate, this is the "ground-truth" data in the sense this is the best

achievable estimate of the normal, and is assumed to be error-free relative to the same calculations performed without de-meaning.

7.3 Effect of Loss of Significance on normal orientation error

7.3.1 Proposed normal error methodology

The normal orientation error is a function of cloud distance from origin and mean point spacing. Thus, to quantify their effect, the B cloud from each of the test datasets was placed at origin and then progressively increased in scale (to increase the mean point spacing) and moved further from the origin. When translating the cloud away from the origin, the whole cloud was shifted equally along all three axes to increase the magnitude of the point coordinates. Note that this methodology deliberately only uses the "B" cloud, ignoring the "A" cloud.

Prior to this, a number of M randomly chosen points were picked from each cloud. At each new scale and distance, each of the selected points had their mean point spacing ($\overline{s}_i$), Euclidean distance from origin ($\|\boldsymbol{p}_i\|_2$) and normal orientation error (θ_i) calculated as described in Section 7.2.2. In choosing a value for the sample size M, 10% of the cloud size was used. This was done to reduce computation times.

A k-nearest neighbor search was used to identify the set of neighbors Q_i closest to each point. In the choice of k, a value of 15 was selected as this is a value commonly used by existing studies [71, 77]. As long as k is within an appropriate range, its exact value is not critical as research has shown that result of the LSQ normal calculation method is consistent for many similar values of k [71, 72].

This experiment was conducted once with the data type set to *float*, and again with it set to *double*. Both sets of results are shown in Figures 7.4 and 7.5, with *float* results shown in the left-hand sub-figures, and *double* results shown in the right-hand sub-figures.

7.3.2 Normal error results

The results for normal testing on each of the three sample test clouds are shown in Figures 7.4 and 7.5.

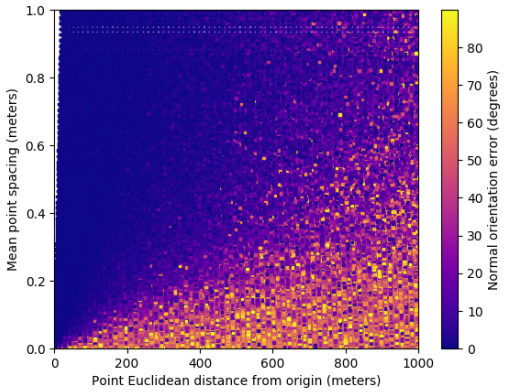
The scatter plots in Figure 7.4 show that, when LoS does occur, the error is highly random, as indicated by the speckled pattern of colors which visually appears to cover the entire range of error from 0° to 90°. A uniformly low error only occurs when the point is close to origin, or has a large mean point spacing. The approximate DDR values at which the mean normal orientation error reaches a variety of thresholds are shown in Tables 7.2 and 7.3.

For all clouds, the mean normal error increases as the DDR increases, which is clearly shown by the mean line in the scatter plots of Figure 7.5. In the left-hand plots, generated using the float data type, the mean orientation error increases rapidly before reaching a stable value of approximately 60° with a wide distribution, as shown by the standard deviation envelope. This result is consistent with the seemingly random orientations shown in Figure 2.8b.

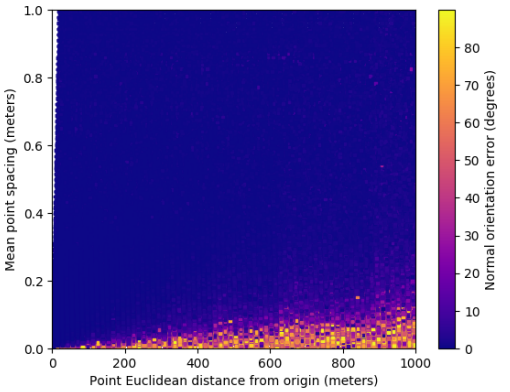
This result is also mathematically consistent because a set of random vectors will have a mean error of exactly 60°. As illustrated in 7.6, if the true normal on a plane is n, then the space of all possible erroneous normal vectors is a hemisphere, where the vector with the mean angle error θ is denoted $\hat{n}$. This vector traces a circumferential line (shown in red) around the hemisphere and if the erroneous normals are *randomly* distributed, then half will fall above the line, and half below it. This line then should also perfectly divide the surface area of the hemisphere, in which case the mean angle can be calculated to be exactly 60°.

The right-hand plots in Figure 7.4 and 8.5(f) were generated using the double data type. So they also show how more significant digits reduc. However, because the number of available significant digits was greater, the degree of LoS was less severe at each distance. As a result, the mean error increases slowly, and does not plateau until a much high DDR, off the end of the plots.

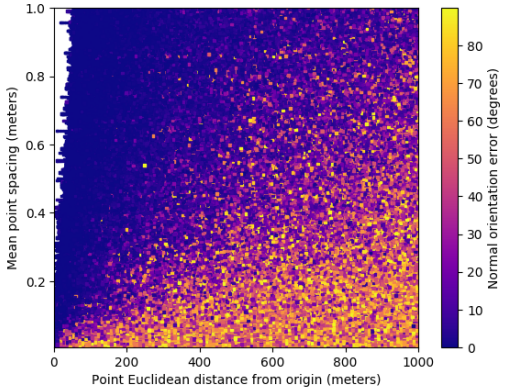
When comparing datasets, the results reflect the nature of the cloud structure. For the Stanford Bunny, the normal error increases slowly, where the DDR must pass 420



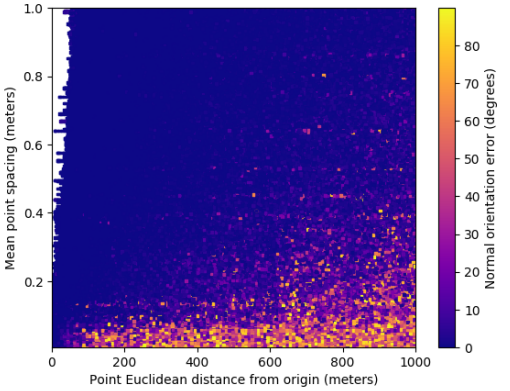
(a) Stanford Bunny (float)



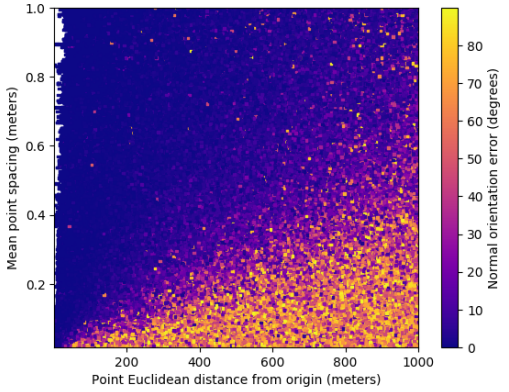
(b) Stanford Bunny (double)



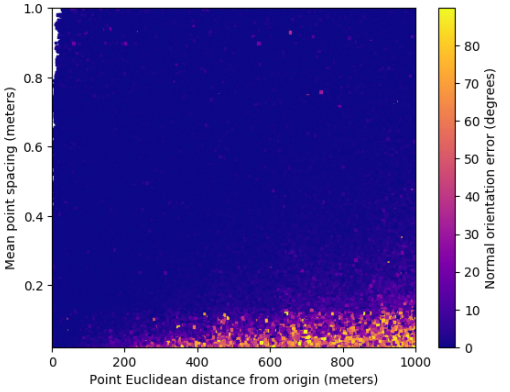
(c) HDL-32 carpark LiDAR scan (float)



(d) HDL-32 carpark LiDAR scan (double)

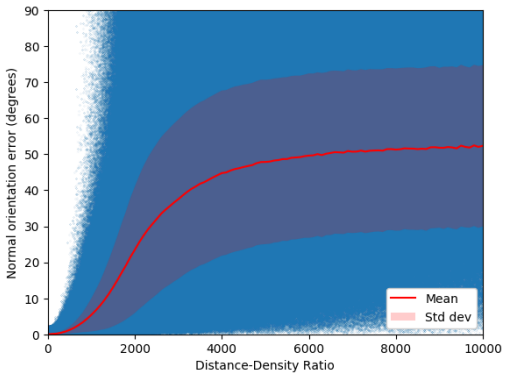


(e) SX10 clock tower LiDAR scan (float)

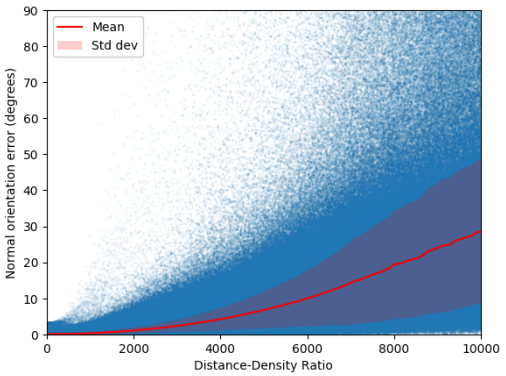


(f) SX10 clock tower LiDAR scan (double)

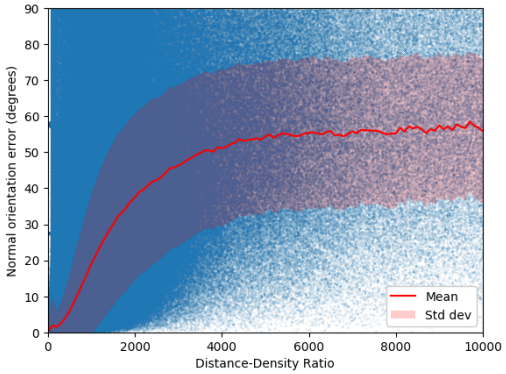
FIGURE 7.4: Normal deviation results. Scatter plots illustrate how loss of significance and normal error is a function of point spacing and point coordinate magnitude. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, plots on the right with the double data type.



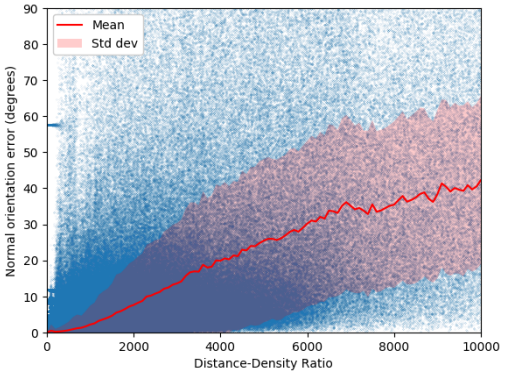
(a) Stanford Bunny (float)



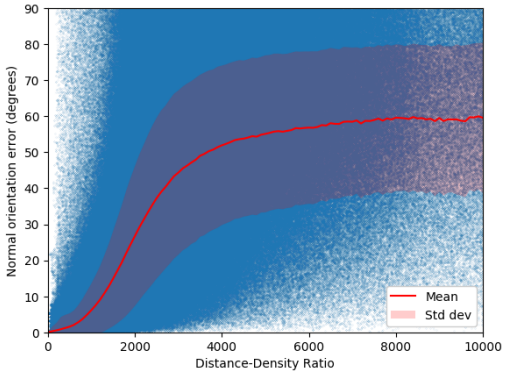
(b) Stanford Bunny (double)



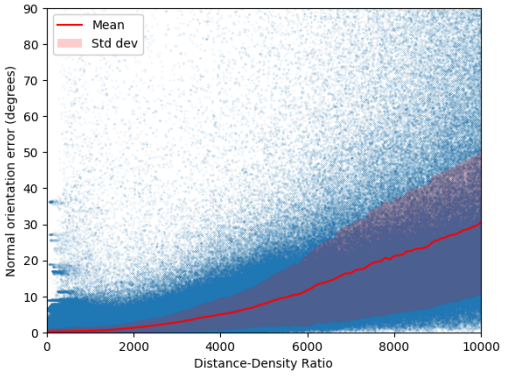
(c) HDL-32 carpark LiDAR scan (float)



(d) HDL-32 carpark LiDAR scan (double)



(e) SX10 clock tower LiDAR scan (float)



(f) SX10 clock tower LiDAR scan (double)

FIGURE 7.5: Normal error mean and standard deviation results. Scatter plots illustrate how as the DDR increases, so does the mean normal deviation from the ground truth. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, plots on the right with the double data type.

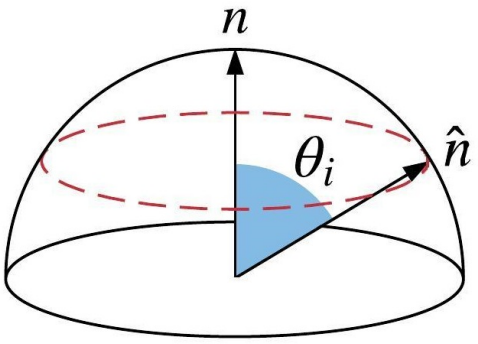


FIGURE 7.6: Diagram illustrating the expected mean error angle of a random normal vector

TABLE 7.2: Distance-Density Ratio at which the mean normal orientation error passes X° when calculated with floats.

	X°		
Cloud	**1°**	**5°**	**10°**
Stanford Bunny	420	930	1290
HDL-32 carpark scan	60	410	620
SX10 clock tower scan	320	860	1220

TABLE 7.3: Distance-Density Ratio at which the mean normal orientation error passes X° when calculated with doubles.

	X°		
Cloud	**1°**	**5°**	**10°**
Stanford Bunny	1890	4320	6000
HDL-32 carpark scan	600	1530	2250
SX10 clock tower scan	1740	3940	5570

before seeing even 1° of error in normal orientation. This means that the distance from any point in the bunny to the origin of the coordinate system must be 420 times the mean spacing between the point and its nearest neighbors before 1° of error will occur in normal calculation. Most point clouds such as the Stanford Bunny are independent models or part of a scene, such as a depth image from a stereo camera. Since such clouds are typically centered at origin or processed in a small reference frame, it is unlikely that users in these types of applications will experience significant LoS. By contrast, Velodyne HDL-32 and Trimble SX10 are often used in geoscience and mapping applications, thus their point clouds are much larger in size with earth-referenced coordinates, often on the order of several hundred or thousand meters.

For the HDL-32 point cloud, the normal error is more pronounced and occurs much

sooner, showing a mean of 1° at just 60 DDR. This is particularly concerning because the HDL-32 (and many similar LiDAR units) generate points with mean spacing on the order of a few centimeters or even millimeters. If the cloud is close to the origin, the normal error may be significant. In some extreme cases, if point density is very tight, and on the order of fractions of a millimeter, normal orientation error may occur within a cloud even if its centroid is at origin.

The particular vulnerability of the HDL-32 scan to normal error is partly because of the unique way in which such 3D spinning LiDAR units work. Because they consist of several single-beam LiDAR units mounted along a spinning vertical axis, the point clouds they produce tend to bunch points together in bands (see Figure 7.3(b)), where each band corresponds to a different beam from the LiDAR. This irregular yet potentially tight spacing along the curve of the band is what is likely contributing to the randomized error seen in all parts of the scatter plot in Figure 7.4c,d.

The results of the SX10 clock tower scan are very similar to the results of the Stanford Bunny, which is intuitive as both clouds predominantly consist of flat or smooth surfaces. If the SX10 clock tower scans contained more organic objects such as trees or bushes, it is likely that the results would be closer to what is observed in the HDL-32 carpark scan results.

With every dataset, results calculated with the double data type clearly show that increasing the number of available significant digits reduces but does not eliminate the level of error caused by LoS. Using data types larger than 64-bit will have the same effect. For most point clouds without very tightly spaced points, the best way to eliminate LoS is to de-mean the point and its nearest neighbors before computing the normal.

7.4 Effect of Loss of Significance on Cloud Registration

7.4.1 Proposed Registration Error Methodology

The final objective of this chapter is to demonstrate the effect of LoS on cloud registration, which is ultimately what will concern most users of PCL. To do this, a similar process was used to the one outlined in Section 7.3.1. A few key differences were because

while the DDR can be calculated on a per-point basis, it cannot be meaningfully calculated for a whole cloud, especially if that cloud has a large variation in point spacing. In addition, the registration error can only be expressed per-cloud.

To quantify the effect of LoS on cloud registration, each dataset was progressively shifted from origin, without altering its scale. Both clouds (A and B) in each dataset were moved as one so that their relative position was maintained. At each increment from the origin, a copy of cloud B was made (B$'$) and translated by a fixed offset to simulate added error. The initial error was chosen to be 0.001 for the Stanford Bunny dataset, and 0.2 for the other datasets. Cloud B$'$ was then registered to cloud A using each described variant of the ICP algorithm. The PICP algorithm was used twice, once using normals calculated in-place, and again with the normals correctly calculated using de-meaning. The mean point-to-point distance of the registered cloud B$'$ and B to the ground-truth was then calculated as the registration error, as described in Section 7.2.2.

Parameter tuning is an important part of cloud registration, as many registration algorithms (ICP variants included) may perform poorly without appropriate tuning. However, to re-iterate, one of the stated objectives of this chapter is not to demonstrate how accurate each ICP method can be, but simply that their accuracy is affected by LoS. Thus, tuning each ICP variant for each dataset is beyond the scope of this thesis. In addition, it would be unfeasible to provide tuned parameters for each ICP variant and dataset combination used in this chapter. Thus, all ICP variants used their default parameters, as determined by the PCL code base. The only exception is that when registering the Stanford Bunny dataset, the maximum correspondence distance was set to 0.002.

Like the normal orientation results, the registration results were calculated with both float and double data types. ICP and PICP use floats by default, however PCL's implementation of GICP is hard-coded to use doubles, and unlike the ICP and PICP implementations is not templated to allow the user to specify the data type. Thus, GICP results are only shown on the plots explicitly labelled "double".

In addition, note that PCL provides several implementations of point-to-plane ICP, this chapter uses the Linear-Least Squares version as described by Low [144].

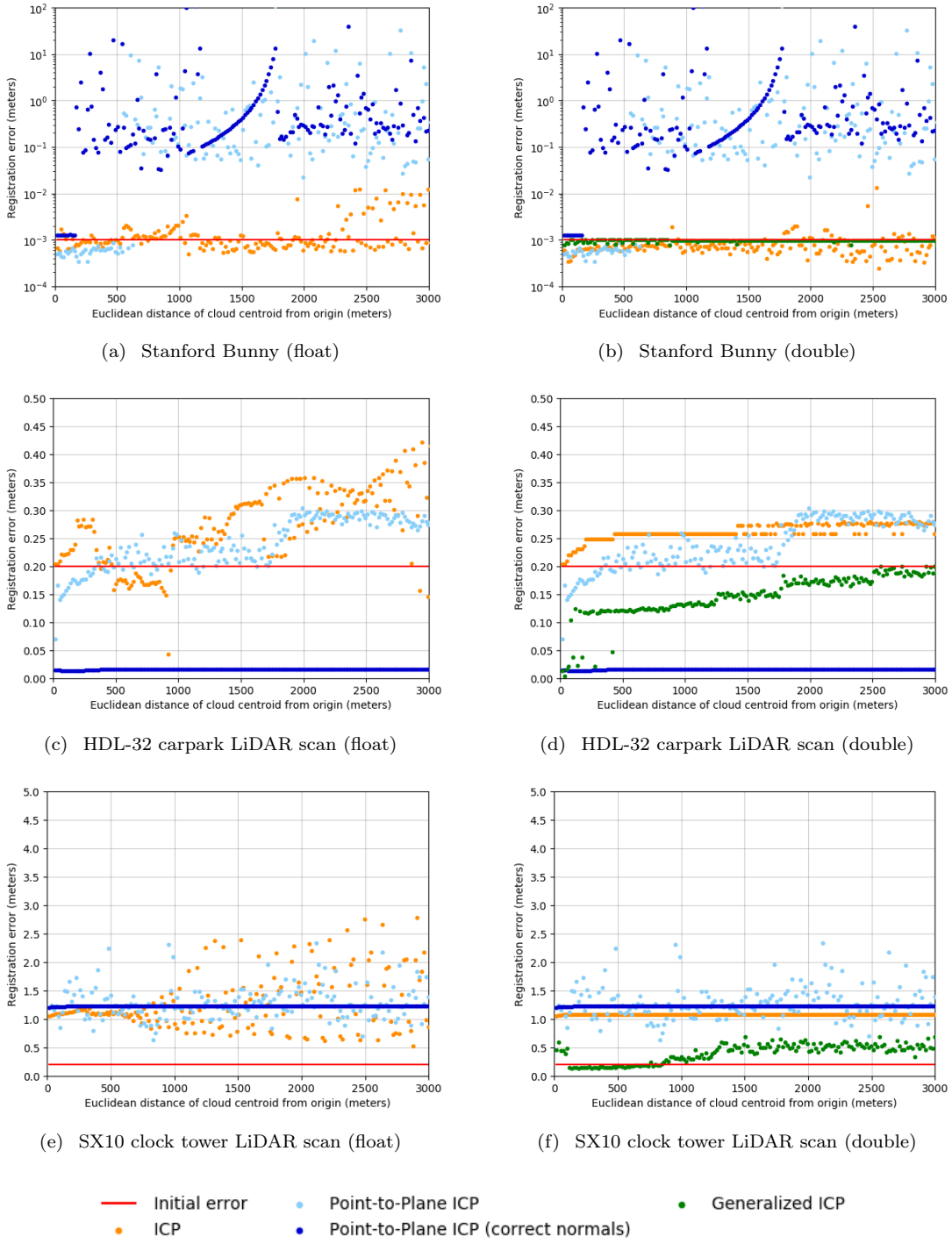
7.4.2 Registration Error Results

The results showing the effects of LoS on cloud registration are provided in Figure 7.7. While these results are subjective as they apply only to PCL's specific implementation of each ICP algorithm, they highlight how unpredictably the algorithms can behave when LoS starts occurring.

The standard ICP algorithm logically performs poorly with all three datasets. Because it simply tries to minimize point-to-point distance, it will never correctly match non-identical clouds. For each cloud pair, there will be some orientation that minimizes this distance yet incorrectly aligns them. When ICP is used to register all three datasets, it converges in an unstable manner, producing inconsistent errors that grow larger as LoS increases. When ICP is used with a double data type instead of float, its result is more consistent although still often worse than the initial error.

The registration results of the PICP algorithm show that its accuracy is solely determined by whether or not the point normals were calculated accurately. The standard PICP results of all three datasets are inaccurate and imprecise regardless of whether they are calculated using floats or doubles. By comparison, when the PICP algorithm is used with correctly calculated normals, it provides precise results at every distance, and regardless of data type. This implies that correct normal calculation can make PICP impervious to LoS. However, during registration of the Stanford Bunny dataset, past approximately 200 meters the results become extremely inaccurate. This is because past 200 meters LoS causes correspondence detection to fail, and therefore the failure of the whole registration algorithm. The end result is the B′ cloud being transformed to a position far away from the A cloud.

In most cases, GICP produces results that are more accurate if not more precise than ICP or PICP. However, it is clear that GICP is still being affected by LoS, as evident by fact that the error increases with distance from origin. It has already been stated that PCL's implementation of GICP calculates the required covariance matrices internally, and explicitly casts them to the double data type. However, there are still some components of the GICP implementation that are hard-coded to use floats, so it is possible that LoS is occurring elsewhere in the GICP process. This would explain why the GICP



(a) Stanford Bunny (float)

(b) Stanford Bunny (double)

(c) HDL-32 carpark LiDAR scan (float)

(d) HDL-32 carpark LiDAR scan (double)

(e) SX10 clock tower LiDAR scan (float)

(f) SX10 clock tower LiDAR scan (double)

FIGURE 7.7: Registration point-to-point error. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, images on the right with the double data type.

results appear to be less precise than PICP with correct normals. Unfortunately, confirming this hypothesis is outside the scope of this research, but a deeper investigation of where exactly LoS occurs in different registration algorithms would make for valuable future research.

Regardless of the flaws of each ICP variant, all perform best when their clouds are very close to, or at the origin. As with the normal orientation results, increasing the size of the data type reduces the problem of LoS but does not eliminate it. Thus, the only way to eliminate the error caused by LoS is to de-mean both clouds prior to registration.

7.5 Best Practices for Avoiding Loss of Significance

Users of PCL as well as any other point cloud processing application will obviously want to avoid issues caused by loss of significance. This chapter demonstrates its effects on normal orientation and cloud registration, however there are many other point cloud processes that can be affected.

The results of this chapter prove that loss of significance can be avoided entirely by centering each point cloud at origin before operating on it. However, this is not always possible, especially in mapping and geoscience applications, where large Earth-referenced point clouds may be required. To this end, the authors suggest the following "best practices" when writing or using functions that deal with point clouds:

1. De-mean each cloud to center it at or close to the origin before operating on it. Some applications such as CloudCompare will explicitly ask the user to do this before loading a cloud with large coordinates.

2. Use or write functions which provide support for larger data types. For example, Libpointmatcher and some PCL functions are templated to allow the user to specify the data type as either float or double.

3. Explicitly check for conditions that indicate potential LoS (e.g., high DDR, large coordinates, and large numbers of significant digits in the result) and warn the user.

4. When dealing with LiDAR or other range data, it may be appropriate to store and process the cloud in the original reference frame of the sensor, and store its local or global coordinate offset separately.

5. In applications that use point normals, the authors recommend visualizing each point cloud and its normal vectors to confirm that they are orientated correctly. The `PCLVisualizer` API can be used for this purpose.

7.6 Summary of Numerical Imprecision and Mapping

This research has demonstrated that loss of significance (LoS) can be a large source of error for cloud registration and point normal calculation. The LoS in this context is a function of the cloud distance from origin and the mean point spacing.

When LoS occurs, it can result in highly random normal orientations. The degree of loss of significance, and the level of error that it generates varies depending on the cloud and the data type used. For a cloud with regular point spacing, such as the Stanford Bunny, the distance of any point from the origin must be 300–400 times the mean point spacing before 1° of error will occur in the point normal orientation. For irregular clouds such as those from a 3D LiDAR unit, this ratio can be as low as 60.

When LoS affects cloud registration, it can result in a sub-optimal match that is potentially several meters or degrees off from the ideal match. It can even cause a subcomponent of the algorithm, such as correspondence detection, to fail completely. The Stanford Bunny results show that, when this happens, it can cause the algorithm as a whole to fail.

The experimental results provided show that the problem can be largely avoided by processing point clouds while they are close to or at the origin of the local coordinate system. Several best-practice recommendations have also been proposed, which readers may find useful for avoiding this problem.

8 | Accurate Registration with Sparse Point Clouds

This chapter describes three distinct experiments. The first compares GICP with mesh-GICP (MGICP) at close to long range (>20 m), beyond the ranges shown in [70, 86, 87]. The objective of this experiment is to demonstrate that MGICP is not only more accurate at matching clouds that are far apart, but also that it is more *reliable*.

The second experiment tests the the point cloud aggregation methods discussed in Section 4.2.4, after each target/source cloud pair has been registered with MGICP. This experiment determines the initial position of each source cloud based only on the registration result of the previous source cloud. This is done to simulate a Visual-Odometry (VO) style mapping system.

The third experiments repeats the second, except that the initial position of each source cloud is obtained by adding random position and rotation error to the ground-truth position of that cloud. This simulates a SLAM style mapping system where other sensor data such as odometry is used to determine the initial position of a cloud.

The objective of the second and third experiment was to determine what combination of point cloud registration, aggregation and initial pose placement methods produces the most accurate combined point cloud. I.e. what combination of methods are best for LiDAR mapping.

All experiments use the same three data sets, which consist of a number of Velodyne HDL-32 clouds, and one cloud of the same area collected using a Trimble SX10 total station as a ground-truth data set. All experiments also use the same setup and error metrics, which are all described in the following sections.

For these experiments, the second robotics prototype, as discussed in Section 5.1.4, was used. Software technical specifications are provided in Appendix A.

8.1 Experimental Setup

8.1.1 Source of Ground Truth

The error metrics and methodology of this research rely on knowing the "ground-truth" position of every cloud in each data set. To calculate this, each cloud was registered to a highly accurate "reference map" collected using a Trimble SX10 total station. This is the method proposed in Sprickerhof *et al.* [145], and similar to what is used in Wulf *et al.* [94]. Each cloud has been aligned with the reference map using a combination of manual positioning and point-to-point ICP in CloudCompare.

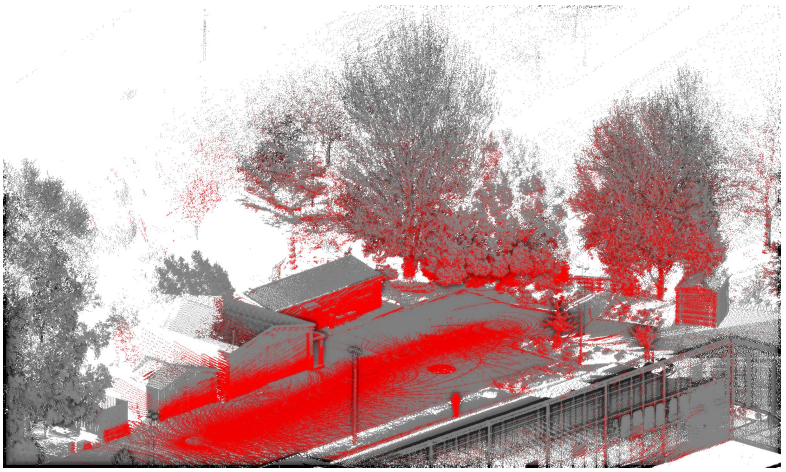
This process converts the coordinates of each cloud to be relative to the reference frame that the SX10 cloud is expressed in, which is an arbitrary local reference frame. But as discussed in Chapter 7, registration conducted when the clouds are far from the origin can cause numerical loss of significance, which can signficantly degrade the performance of the algorithm. To avoid this, every cloud is de-meaned such that the first cloud in the sequence becomes the origin of the coordinate system. I.e. the coordinates of every cloud after the first is transformed to be relative to the first.

The data sets are shown in Figure 8.1, where the reference map is shown in grey, and the clouds in red.

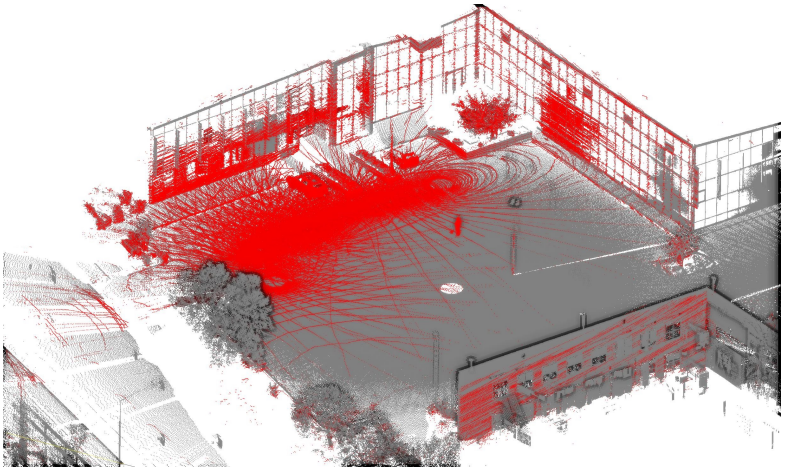
8.1.2 Error Metrics

Each data set used in this research consists of n clouds and their associated poses $\{A_i,\ \mathbf{P}_i \mid i \in 1 \ldots n\}$, where each pose marks the position and orientation of the sensor (the HDL-32 LiDAR unit) relative to an arbitrary global coordinate system. Poses are represented by a 4x4 transformation matrix consisting of two elements: a rotation $\mathbf{R}_i$ and translation vector $\mathbf{t}_i$. Since the clouds are in their ground-truth positions, the poses are denoted $\mathbf{P}_i^{tru}$.

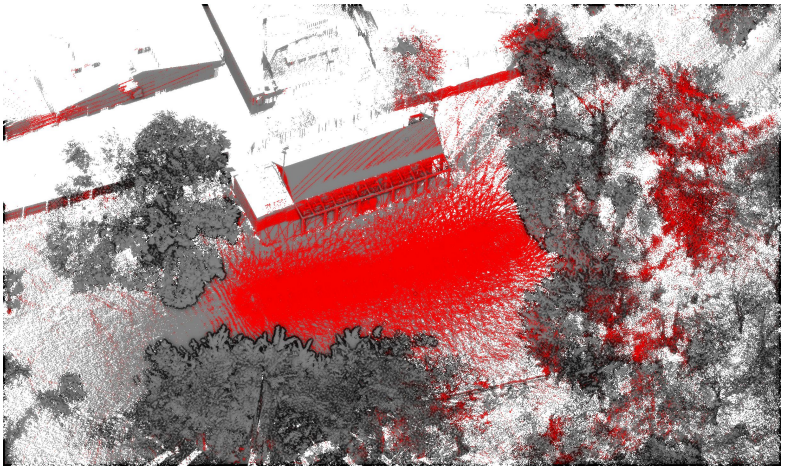
With reference to Figure 8.2, most robotic systems model the robots trajectory as a series of single poses ($\mathbf{P}_i,\ \mathbf{P}_{i+1},\ \mathbf{P}_{i+2}, \ldots$). In reality, any given pose $\mathbf{P}_i$ can have many values (i.e. positions and orientations). If a full SLAM solution is implemented, the initial placement of the next cloud (denoted $\mathbf{P}_i^{est}$) is based on the prior registration result as



(a) Data set 1: Garage

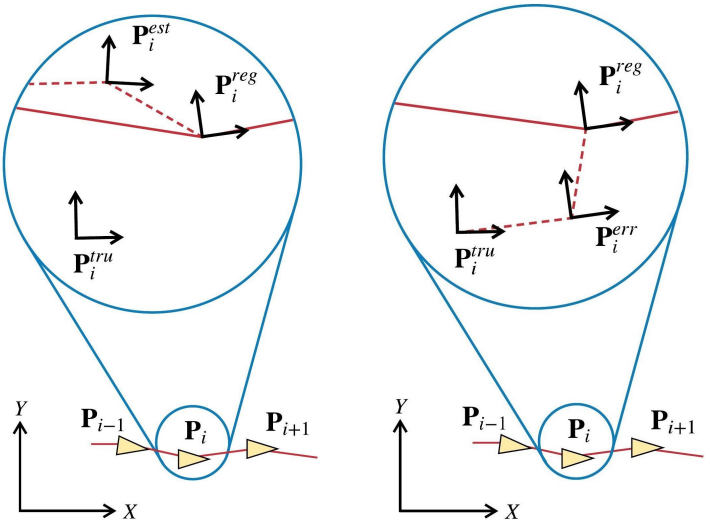


(b) Data set 2: Carpark



(c) Data set 3: Forest

FIGURE 8.1: Data sets used in this research. Grey points come from a Trimble SX10 scanning total station, red points come from one of several ground-truth HDL-32 scans that has been registered to the SX10 cloud.



(a) VO or SLAM-style pose representation

(b) Mapping-style pose representation

FIGURE 8.2: 2D example of reference frame notation used in this research. Solid red lines represent the final trajectory, while dashed red lines represent the links between related poses. Each pose is given relative to a common coordinate system.

well as other sensor data. If the system uses VO, then only the prior registration result is used. In both cases the estimate can drift progressively further from its true value $\mathbf{P}_i^{tru}$. This model is illustrated in Figure 8.2a.

Alternatively, when mapping applications use absolute positioning system such as GNSS, the error from previous registration does not accumulate. In this case it is more appropriate to model each estimated pose as an error-added copy of its true position $\mathbf{P}_i^{err}$, as shown in Figure 8.2b.

Depending on the experiment and methodology used, the estimated or error-added cloud is registered to the previous cloud. The result is a cloud and pose that have been transformed by the output of Equation 4.5, e.g. $\mathbf{P}_i^{reg} = \mathbf{T}_{GICP}\mathbf{P}_i^{err}$ *or* $\mathbf{P}_i^{reg} = \mathbf{T}_{GICP}\mathbf{P}_i^{est}$

In all experiments in this chapter, the registration error is defined as the difference between a clouds true $\mathbf{P}_i^{tru}$ and registered $\mathbf{P}_i^{reg}$ position. Specifically the transform that expresses the registered pose relative to the true pose, which is defined as follows:

$$\mathbf{E}_i = \begin{bmatrix} \mathbf{R}_i^E & \mathbf{t}_i^E \\ 000 & 1 \end{bmatrix} = (\mathbf{P}_i^{tru})^{-1}\mathbf{P}_i^{reg} \tag{8.1}$$

The translation component of this matrix is then just the Euclidean distance between them, which is the same translation error metric used in other studies [94, 141, 145]:

$$e_i^t = |\mathbf{t}_i^{tru} - \mathbf{t}_i^{reg}| = |\mathbf{t}_i^E| \tag{8.2}$$

When quantifying the rotation error, we use the metric proposed by [141], which is the angle expressed by an axis-angle representation of the error transform rotation matrix:

$$e_i^r = arccos\left(\frac{trace(\mathbf{R}_i^E) - 1}{2}\right) \tag{8.3}$$

Also like [141], we calculate the median value where multiple data points are collected. The median is more robust to outliers than alternative metrics like the mean or root-mean-square. And as the results in Section 8.2.1 show, the unmodified GICP algorithm is prone to producing outliers when given a poor estimate for the initial alignment.

The error metrics above quantify how accurate any given registration operation is. But they do not indicate how long that method remains accurate in the case where different aggregation methods are compared. The simplest way to quantify this is to set a threshold, and identify when the positional accuracy of a method exceeds it.

For this research, we set the positional accuracy threshold at 0.25 m, as this is the lowest horizontal accuracy of the R7 GNSS system (when in differential mode). We set the rotational threshold at 1.5° as this is the lowest stated accuracy. These thresholds then indicate the point beyond which it would be more accurate to use raw sensor data to localize the clouds, rather than registration. A position error threshold of 0.25 m is also the same as the "strict" threshold used in [86].

8.2 Experiments

8.2.1 Experiment 1: GICP vs MGICP

To compare the accuracy of GICP vs MGICP, both algorithm were used to sequentially register each cloud in each data set using the keyscan aggregation method. For every cloud A_i in a set of n clouds $\{A_i \mid i \in 2...n\}$, A_i was registered to the first cloud in the set $A_k,\ k = 1$ (the "key" scan).

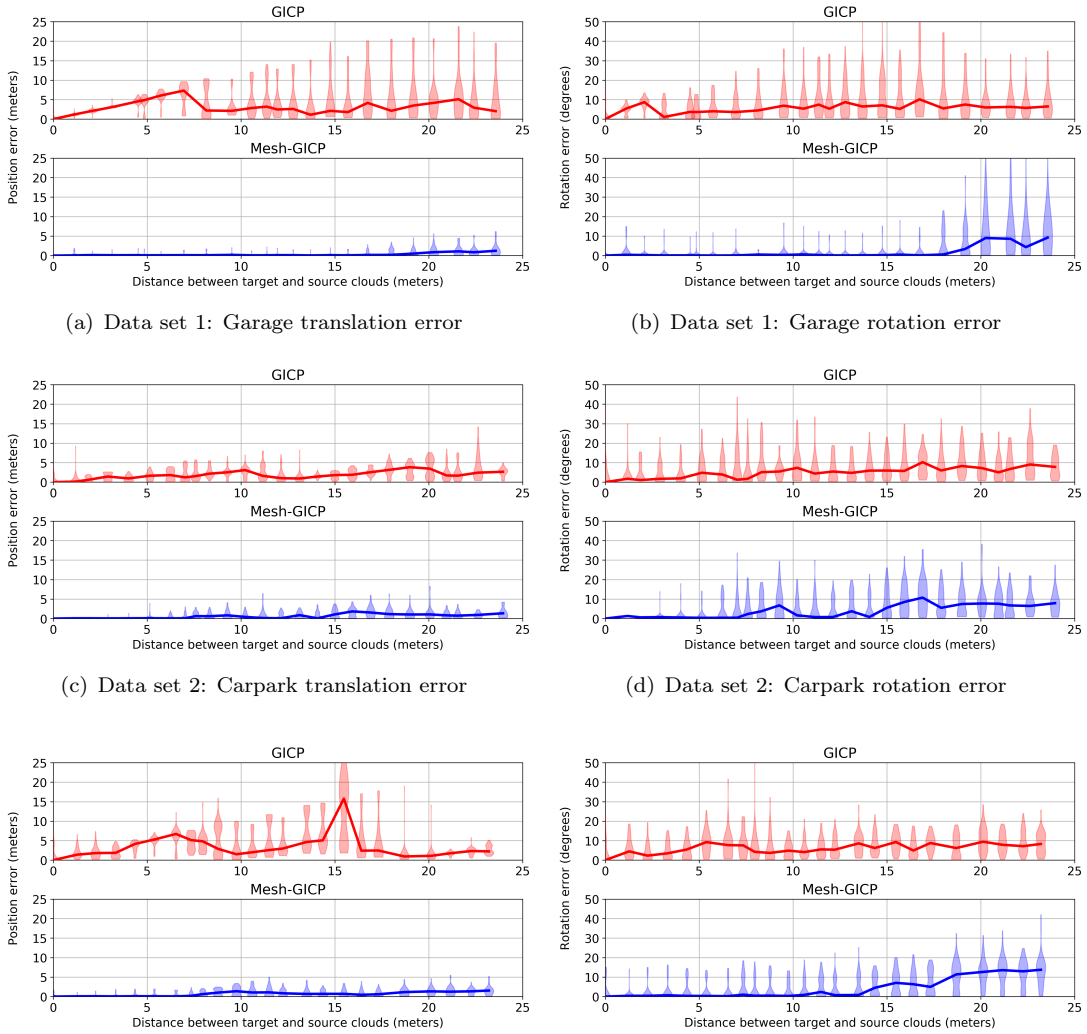
For testing, given that all the clouds were already aligned, uniform random error was added first. So for each cloud pair, a copy of the source cloud A_i was made. Then error was added to it and it's corresponding pose (now $\mathbf{P}_i^{err}$). The error-added cloud was then registered to generate $\mathbf{P}_i^{reg}$ as per Figure 8.2a. Following the methodology in [82], the error was set to a uniformly distributed translation and rotation with bounds of ± 1.5 m and $\pm 15°$ respectively.

The translation and rotation error of the registered result were computed as per Section 8.1.2. This process was repeated 50 times for each cloud, with a freshly generated error each time. The distribution of the results form the violins shown in Figure 8.3. The median translation and rotation error for each set of cloud pair results were also calculated and plotted as a line. Note that the X-axis of Figure 8.3 is the distance between target (the first cloud in the set, also the keyscan) and source cloud. So each subsequent cloud in each data set starts further from the target cloud than the last (before error is added) and therefore has less overlap with target cloud.

8.2.2 Experiment 2: VO-style Registration with MGICP

To compare the effectiveness of each aggregation method, MGICP alone was used. Each aggregation method is already described in Section 4.2.4, we now apply the $^{tru}, ^{err}, ^{reg}$ notation to more formally describe how each method was implemented in this test:

1. **Pairwise:** $A_i^{est} \Rightarrow A_{i-1}^{reg}$.
2. **Metascan:** $A_i^{est} \Rightarrow M = \{A_j^{reg} \mid j \in 1 \ldots i-1\},\ j > 1$.
3. **Keyscan:** $A_i^{est} \Rightarrow A_k^{tru},\ \{A_i^{est} \mid i \in 2 \ldots n\},\ k = 1$.



(a) Data set 1: Garage translation error

(b) Data set 1: Garage rotation error

(c) Data set 2: Carpark translation error

(d) Data set 2: Carpark rotation error

(e) Data set 3: Forest translation error

(f) Data set 3: Forest rotation error

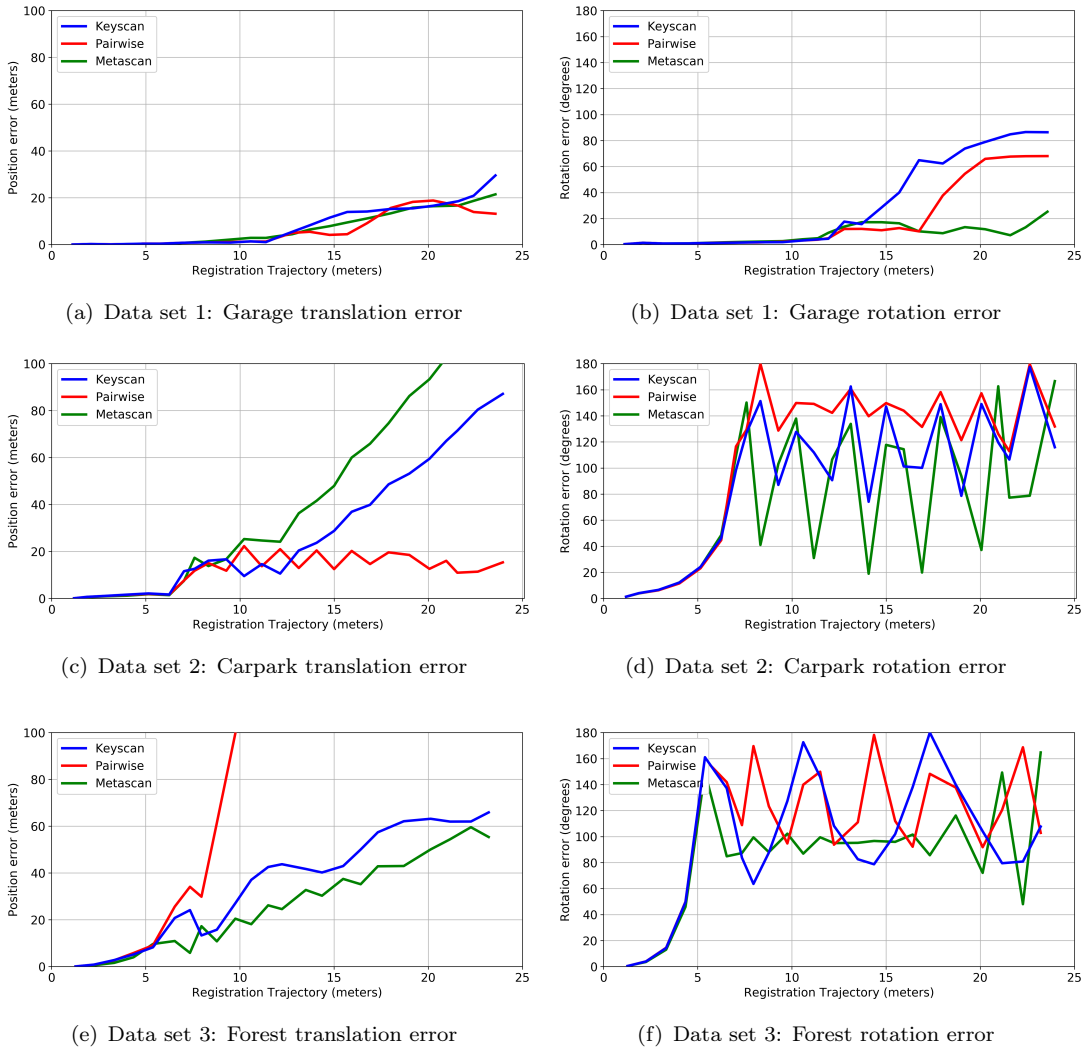
FIGURE 8.3: Experiment 1 results: Violin plots showing the GICP vs MGICP error. Each violin shows the distribution of 50 registration operations. GICP results are red, MGICP are blue. Plotted lines indicate the calculated median error.

This experiment follows the unconstrained VO-style pose system, where the initial placement of each pose is determined by the previous registration. This allowed the registered cloud to deviate from its true position without bound.

Each initial registration of the first cloud was always to its ground-truth version. I.e. $A_2^{err} \Rightarrow A_1^{tru}$ And again, as in the previous section, for each cloud the error was calculated between $\mathbf{P}_i^{err}$ and $\mathbf{P}_i^{tru}$. The results of this test are shown in Figure 8.4.

Note that point cloud registration is an entirely deterministic operation. So as this test did not add randomly generated error, the results for each registration operation are

identical. This is why Figure 8.4 shows only the position and rotation error lines and not a full violin graph.



(a) Data set 1: Garage translation error

(b) Data set 1: Garage rotation error

(c) Data set 2: Carpark translation error

(d) Data set 2: Carpark rotation error

(e) Data set 3: Forest translation error

(f) Data set 3: Forest rotation error

FIGURE 8.4: Experiment 2 results: Aggregation error when using VO-style estimated initial positions. Note that no violins are shown as this is an entirely deterministic test.

8.2.3 Experiment 3: Mapping-style Registration with MGICP

The third experiment was similar to the second, except that it used mapping-style poses. So that instead of relying on the previous registration, the initial pose of each cloud was determined by adding random error to the ground-truth cloud. This simulates a SLAM-style system where the initial pose of a cloud may be determined by sensor data such as odometry. Again the aggregation methods are listed here for clarity:

1. **Pairwise:** $A_i^{err} \Rightarrow A_{i-1}^{reg}$.
2. **Metascan:** $A_i^{err} \Rightarrow M = \{A_j^{reg} \mid j \in 1 \ldots i-1\},\ j > 1$.
3. **Keyscan:** $A_i^{err} \Rightarrow A_k^{tru},\ \{A_i^{err} \mid i \in 2 \ldots n\},\ k = 1$.

Because the error can be determined randomly, multiple samples were collected. Each sample consisted of one complete registration trajectory, i.e. one complete sequence of registration operations from cloud A_0 to cloud A_n. Every creation of pose $\mathbf{P}_i^{err}$ was done with a freshly generated uniform random error. As with Experiment 1, 50 samples were collected for each aggregation method, and used to generate a violin plot, where each violin represents the error distribution for that source cloud (Figure 8.5).

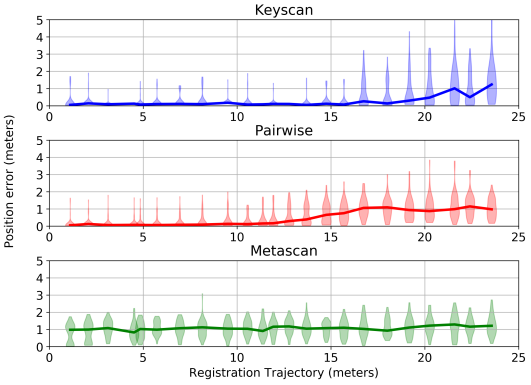
8.3 Discussion

8.3.1 GICP vs MGICP Accuracy

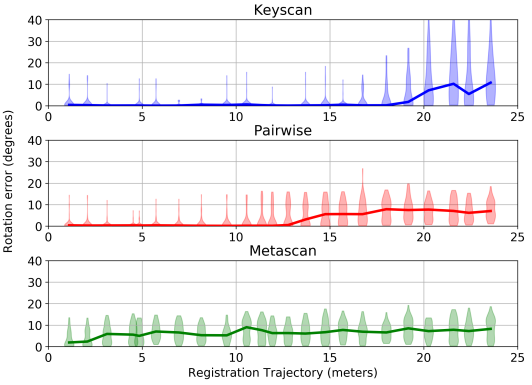
The results of the GICP vs MGICP experiment expand on the work of [70, 86, 87] by showing that MGICP is superior to GICP at a wide range of distances between the source and target cloud. MGICP outperforms GICP in both position and rotation accuracy. By collecting multiple samples with different error, the results also clearly highlight that MGICP is also significantly more robust to error. As the results in Figure 8.3 show, not only are the median translation and rotation errors lower for MCIP, but the distributions of the errors are also much tighter and closer to the median. Ordinary GICP by comparison can have a wide distribution, sometimes with clusters of results away from the median.

In addition, the outlying errors are much more extreme, as evident by the maximum height of each violin of GICP data. These outliers indicate that even when the target and source cloud are close together, ordinary GICP can leave the source cloud further from the target than its initial error-added position. Although in most VO or SLAM systems, registered clouds will likely be close together, the distance between source and target clouds is important for the keyscan aggregation method.

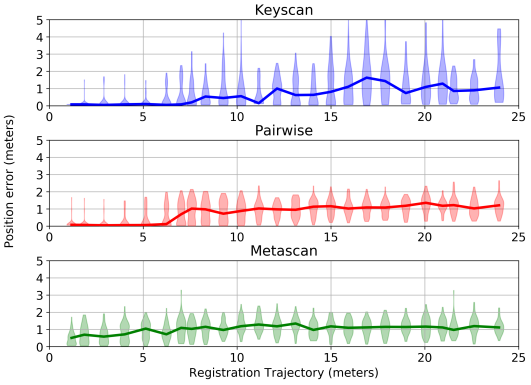
The distances at which the positional accuracy threshold (0.25 m) and rotational accuracy threshold (1.5°) are exceeded by each method are shown in Table 8.1. They clearly



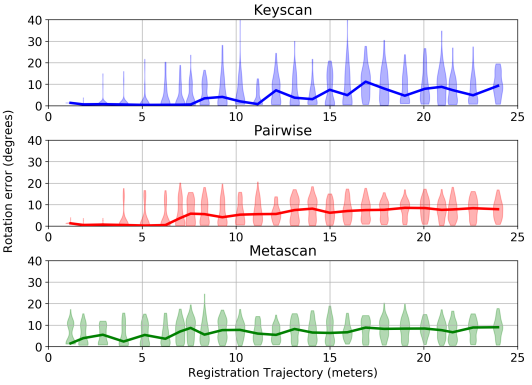
(a) Data set 1: Garage translation error



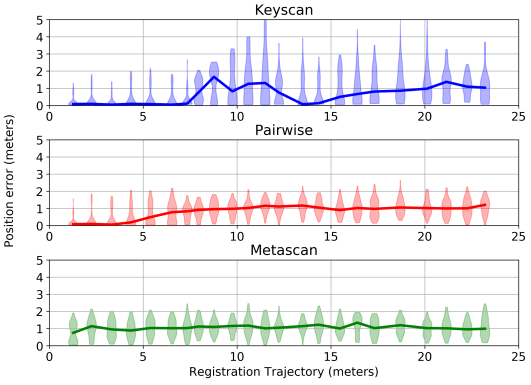
(b) Data set 1: Garage rotation error



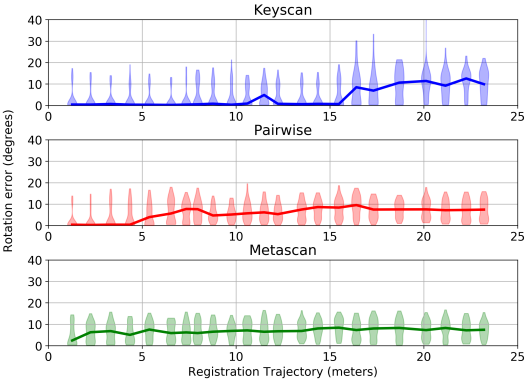
(c) Data set 2: Carpark translation error



(d) Data set 2: Carpark rotation error



(e) Data set 3: Forest translation error



(f) Data set 3: Forest rotation error

FIGURE 8.5: Experiment 3: Aggregation error when using mapping-style estimated initial positions. Each violin shows the distribution of 50 registration operations. Plotted lines indicate the calculated median error.

show that that MGICP remains accurate at distances approximately 4 to 17 times greater than ordinary GICP, which loses significant accuracy after only 1 m between the target and source cloud.

TABLE 8.1: Trajectory length at which the position and rotation thresholds are exceeded for Experiment 1

	Position			Rotation		
Data set	**1**	**2**	**3**	**1**	**2**	**3**
Mesh-GICP	19.18	7.59	7.96	19.18	7.59	11.50
GICP	1.12	1.87	1.28	1.12	1.19	1.28

Position error threshold: 0.25m. Rotation error threshold: 1.5°

8.3.2 Aggregation Method Accuracy

When using VO-style initial pose estimation, the biggest source of error doesn't come from the aggregation methods themselves, but the simple fact that any registration error accumulates. In Wulf *et al.* [94] the position errors can be reduced when the LUM pose graph recognizes a closed loop and adjusts the trajectory. But in this experiment, the registration error accumulates before any definitive differences between aggregation methods can be determined, which limits its ability to show differences between aggregation methods. This is a problem for other comparisons, such as that shown in Wulf *et al.*, and is the primary motivation for conducting experiment 3 where registration error cannot accumulate [94].

When using the mapping-style initial placement of each cloud, the differences between aggregation methods become more apparent. Metascan point cloud aggregation performs the worst in both position and rotation accuracy because only the source cloud is organized, and has covariances computed from a mesh. The target cloud (the metascan) is unorganized and any registration errors accumulate when each new cloud is added to the metascan. This is also why the distribution of errors are more widely spread. Table 8.2 reflects this with the metascan method never achieving an accuracy below the thresholds.

Pairwise aggregation retains the benefit of having both source and target covariances derived from an organized mesh. But because the registered position of each cloud is influenced by the position of the previous cloud, a significant failure in the registration

process compromises the accuracy of every subsequent operation. This is evident from the abrupt increases in error in the median line of the pairwise results in Figure 8.5.

Keyscan aggregation is the most accurate aggregation for mapping because it makes full use of the MGICP algorithm and keeps each registration operation independent. The limitation is that as distance between source and target cloud increases, the amount of overlap decreases. And beyond a certain distance, registration will obviously fail completely. Table 8.2 shows that this distance can be as low as 8 m or as high as 19 m, depending on the data set. This is why keyscan aggregation is not used in SLAM. But in mapping applications where absolute position information is available, keyscan aggregation would be the superior choice for creating accurate local point cloud maps.

TABLE 8.2: Trajectory length at which the position and rotation thresholds are exceeded for Experiment 3

	Position			Rotation		
Data set	**1**	**2**	**3**	**1**	**2**	**3**
Keyscan	16.75	8.33	7.96	19.18	8.33	11.50
Pairwise	12.78	7.03	5.39	13.70	7.03	5.39
Metascan	1.12	1.19	1.28	1.12	1.19	1.28

Position error threshold: 0.25m. Rotation error threshold: 1.5°

8.4 Summary of Registration with Sparse Point Clouds

This research expands the work of [70, 86, 87] by comparing ordinary vs mesh-based GICP with a range of data sets and target/source cloud distances. The position and rotation errors clearly show that MGICP is significantly more accurate *and* more precise at registering sparse point clouds. This is demonstrated in Figure 8.5(f) where the violins for each MGICP data set are significantly shorter than their GICP equivalents, indicating a lower spread of translation and rotation error. The results also show that MGICP is much more successful that GICP at registering sparse point clouds which are far apart.

This research then compares different point cloud aggregation methods (pairwise, metascan and keyscan) with MGICP to determine which is the most accurate at positioning clouds via registration. This comparison is done with two different approaches to initial cloud placement: a VO-type approach where initial placement depends on the previous

registration operation, and a mapping-type approach where initial placement is an error-added deviation from the ground truth. The results of these tests show that accumulated registration error in the SLAM scenario exceeds any differences in accuracy between the aggregation methods. While in the mapping scenario, keyscan aggregation outperforms the other methods in both accuracy and precision up to a given distance between source and target clouds.

9 Accurate Earth-referenced Mapping

9.1 Proposed Method

9.1.1 Overview

Having discussed some of the problems common to many SLAM and mapping systems in Section 4.5, there was clear need for a mapping solution with the following design principles:

1. Discretization of the map into independent sub-maps so as to contain errors that occur during point cloud registration
2. Prioritization of GNSS information above other sensor data and above cloud registration
3. The expectation that significant heading error may be present and so the orientation of individual poses cannot be highly trusted.
4. A system for identifying and removing failed sub-maps.

This chapter proposes a novel method for achieving accurate earth-referenced point cloud maps using a mobile robot. The method is called Anchor Cloud Mapping (ACM), and was inspired by how total stations use calibrated positions and backsight baselines to correct their position and orientation. This same principle was used, along with the lessons learned in Chapters 6 to 8, to develop this novel method. An overview is shown in Figure 9.1, with each core stage of the process numbered. The rest of this section explains the method in greater detail, providing motivation for each design aspect along the way.

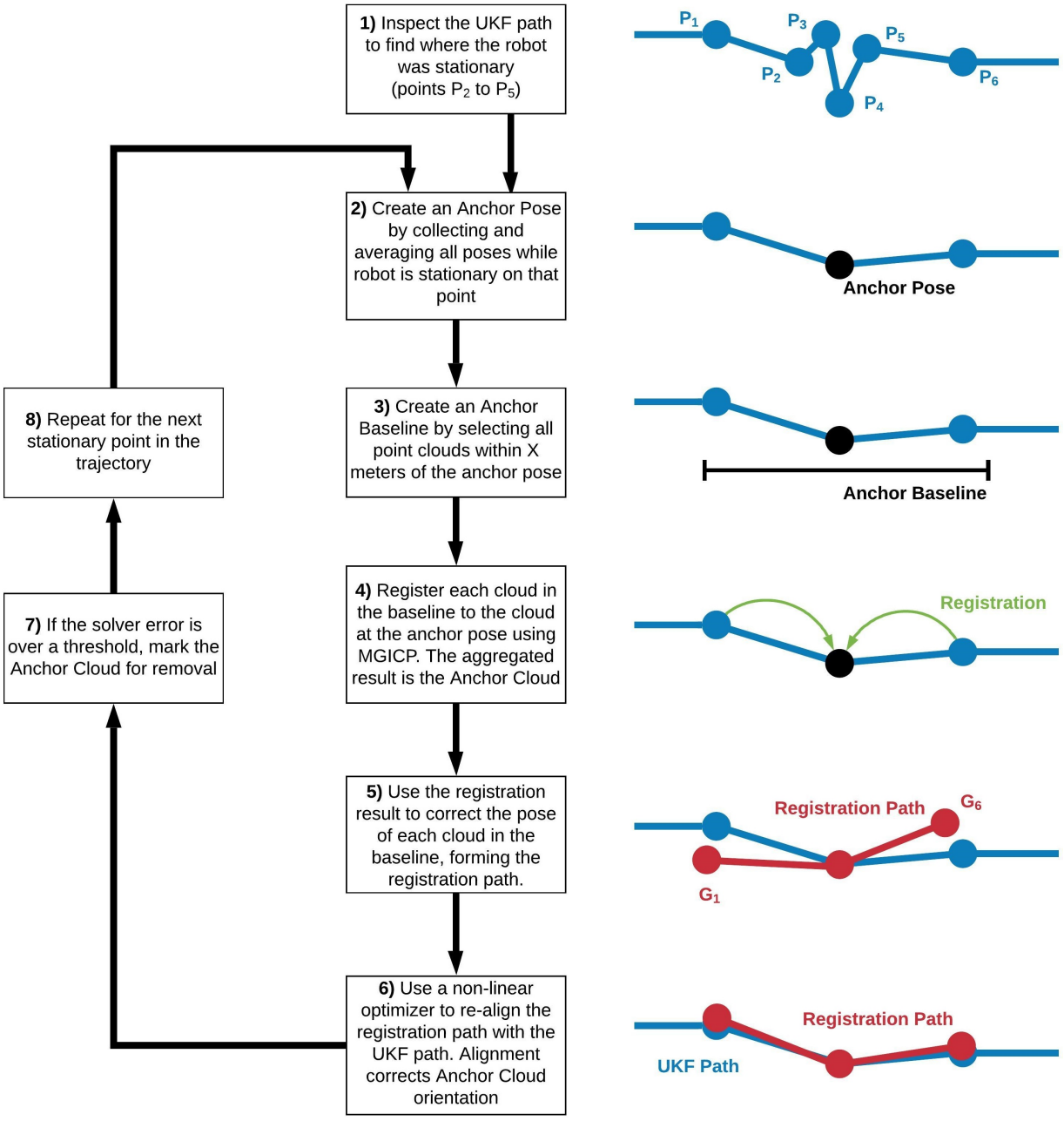


FIGURE 9.1: Anchor Cloud Mapping flow diagram

9.1.2 Anchor Cloud Construction

On the vast majority of GNSS-enabled robots, the GNSS system is the sole source of earth-referenced information. The RTK process has a rated accuracy which can only be improved further by accumulating and averaging many sequential position measurements. This is obviously only possible when the robot is stationary, so any pause in movement is an opportunity for the robot to increase the accuracy of its position information.

Figure 5.6 from Chapter 5 shows a 3 minute sample of elevation data from the robot. In this data, it can clearly be seen that any given elevation measurement can vary by several millimeters between measurements. This data validates the motivation for averaging several sequential measurements to improve accuracy.

The primary source of position information is the Unscented Kalman Filter (UKF) which fuses the GNSS data with IMU and odometry sensor data. It is configured to produce position estimates at 30 Hz, so even a few seconds spent stationary generates multiple position measurements in approximately the same area. In Figure 9.1 these are poses $\mathbf{P}_2$ to $\mathbf{P}_5$. Recall from Section 4.3: State Estimation Filters, that the UKF produces a state which is the *estimated* mean state, where the true state may exist elsewhere within the covariance ellipse, or even outside of it. The implicit assumption is that averaging these poses produces a state which is closer still to the true state, and is therefore the best achievable estimate of where the robot is in earth-referenced coordinates. Strategically stopping the robot at regular intervals creates a series of these poses throughout the environment.

Each averaged pose is then called an "Anchor Pose" as its function is now to anchor the surrounding poses and associated point clouds to the UTM grid. Averaging the position data from n poses is done by averaging the Euclidean x, y and z values of each position. The average orientation is found by converting the rotations to quaternion form and creating a matrix of weighted quaternions $\mathbf{M}$. The average orientation (i.e. mean quaternion $\bar{\mathbf{q}}_i$) is found by first computing the largest eigenvalue, and then the corresponding eigenvector [146]:

$$\mathbf{M} = \sum_{i=1}^{n} w_i \mathbf{q}_i \mathbf{q}_i^T \tag{9.1}$$

$$\bar{\mathbf{q}} \ = \text{argmax } \mathbf{q}^T \mathbf{M} \mathbf{q} \tag{9.2}$$

The next task is to connect the surrounding poses and their associated point clouds with this anchor pose. This is done with the MGICP registration algorithm, using the keyscan aggregation method. The results of Chapter 8 show that this is the most accurate way of registering a group of scans, up to a certain distance between target and source cloud. This suits the goal of ACM to break the complete point cloud map into independent sub-maps. With the anchor pose as the central point, the clouds along the robots trajectory

in either direction are registered to the cloud at the anchor pose, and the complete aggregated cloud is then referred to as the "anchor cloud".

Chapter 7 details how severely cloud registration can be adversely affected when the clouds are far from the origin. So to mitigate this risk, all point clouds in ACM are de-meaned prior to registration.

The empirical results shown in Figure 8.5 suggest that keyscan aggregation starts to lose accuracy past approximately 7 m. So clouds further from the anchor pose than this threshold are excluded. In the example shown in Figure 9.1 this set includes just points $\mathbf{P}_1$ and $\mathbf{P}_6$. The post-registration poses of each cloud are denoted $\mathbf{G}_i$. If there is any rotational error in the anchor pose, the trajectory formed by the post-registration poses may be offset from the original UKF path. The end result is a set of point clouds which are *locally* consistent and accurate in position relative to one another, but as an aggregated cloud have an orientation error relative to the global coordinate system.

The distance between these thresholds, and across the Anchor Pose, is called the "Anchor Baseline" and can be up to 14 m long. This registration path is analogous to a baseline between a total station and a backsight, or to the baseline between two GNSS antenna. They are analogous in the sense that they are all used to correct the orientation of the device in question using lines drawn between GNSS positions. Matching the registration path to the UKF path provides a rotation which can be used to correct the orientation of the whole anchor cloud.

As a final processing step, each anchor cloud is down-sampled after construction using a voxel grid. The points in each voxel are down-sampled by replacing them with their centroid. This research used the `UniformSampling` class provided by PCL, with the voxel cell size set to 0.01, as this is the finest voxel size that can be reliably used without causing the voxel indices to overflow and crash the ACM program. Down-sampling helps manage the size of each anchor cloud, typically reducing it from tens of millions of points to just 2-5 million points. This also produces a more uniform distribution of points so that features close to the robot (like the ground) are not over-represented by point density.

9.1.3 Path Error Calculation

The objective of aligning the registration path with the original UKF path is to produce a rotation that can be applied to the anchor cloud to correct any errors in its orientation. Depending on the initial rotation error of the anchor pose, the discrepancy between the two paths can be negligible or on the order of several degrees.

This process requires more than simply aligning the paths themselves. The orientations of the constituent poses must be respected as well, insofar as that is possible. To rephrase the problem: the goal then is to find a transform $\mathbf{T}$ that aligns the pre-registration poses with their post-registration equivalents by pivoting them about the Anchor Pose. Each path is described as a set of n poses from any segment of the robots total trajectory $P = \{\mathbf{P}_i, \mathbf{P}_{i+1}, ..., \mathbf{P}_{i+n}\}$ and $G = \{\mathbf{G}_i, \mathbf{G}_{i+1}, ..., \mathbf{G}_{i+n}\}$. Both sets of poses are originally given in coordinates relative to the global reference frame, and must first be transformed to coordinates relative to the Anchor Pose. Let these poses be denoted $\mathbf{P}'_i$ and $\mathbf{G}'_i$. The difference to minimize is the delta in rotation and translation between $\mathbf{P}'_i$ and $\mathbf{TG}'_i$, expressed in terms of Euclidean x, y, z and roll, pitch and yaw as follows:

$$\Delta\mathbf{t}_i = [\Delta t_{ix}, \Delta t_{iy}, \Delta t_{iz}]^T \tag{9.3}$$

$$\Delta\boldsymbol{\theta}_i = [\Delta\theta_{ir}, \Delta\theta_{ip}, \Delta\theta_{iy}]^T \tag{9.4}$$

Note that this is slightly different to how the error between point cloud poses has been expressed previously. The comparison of GICP registration and aggregation methods in Chapter 8 uses Equations 8.1 to 8.3, which will be used again later in this chapter for quantifying the error of each Anchor Cloud.

Describing the difference in orientation as a difference in roll, pitch and yaw is more physically intuitive and allows each component to be weighted separately. This is important because as discussed previously in this research, even though the orientation comes from a UKF (section 4.3.3), it's accuracy can be corrupted by poor sensor data from the IMU.

This is most noticeable in the heading for which the UKF is reliant on data from the IMU's magnetometer. The rated heading accuracy for the IMU used in this research is $\pm 1.5°$, although in practice higher heading errors in the UKF poses than this are frequently observed. Recall from the results of Chapter 8 that MGICP with the keyscan

aggregation method achieves a median heading error lower than 1.5° up to the 7m threshold which is used when creating the Anchor Cloud. The assumption is then that the post-registration poses have a better relative orientation accuracy than the original UKF poses. For this reason it is desirable to de-weight the yaw when aligning the paths. Since even a perfectly linear path will also constrain the pitch of the overall path, the pitch can be de-weighted as well.

The weights for the position can be expressed as the scalar value w_{pos}, while the orientation weights can be expressed as:

$$\mathbf{w}_{rot} = [w_{roll}, w_{pitch}, w_{yaw}]^T \tag{9.5}$$

When searching for the optimal transform $\mathbf{T}$ to align the pre- and post-registration poses, it is unnecessary to solve for the 4x4 transform matrix directly. Instead, because the transform only describes a rotation and no translation, it can be expressed as Euler roll, pitch and yaw values $(\theta_{roll}, \theta_{pitch}, \theta_{yaw})$ reduces the solution space to just three dimensions. The transform matrix can then be constructed from these values. The complete error function to be minimized can then be expressed as follows:

$$\text{argmin}\, f(\theta_{roll}, \theta_{pitch}, \theta_{yaw}) = \sum_{i=1}^{n} ||\mathbf{w}_{rot} \Delta\boldsymbol{\theta}_i||^2 + \sum_{i=1}^{n} w_{pos} ||\Delta\mathbf{t}_i||^2 \tag{9.6}$$

As this error function is non-linear, an appropriate non-linear optimizer can be used to solve it. For this, the open-source NLopt [147] library was used. Also note that the result of this equation is unit-less.

9.1.4 Path Non-linear Optimization

NLopt offers several different optimization algorithms than can be implemented. These algorithms are divided into categories based on whether they are global or local optimizers, and whether they are gradient-based or not.

Given that the pre and post registration paths should only deviate by a small angle, the optimal rotation should be near in the search space to the starting values. For this reason, and because the NLopt documentation states that "*Many of the global optimization algorithms devote more effort to searching the global parameter space than in finding*

the precise position of the local optimum accurately" [148], a local search algorithm can be expected to find a more accurate solution faster than a global search algorithm. Several such NLopt algorithms were selected and tested to find the optimal choice for this function. Specifically:

- Principal Axis (PRAXIS) [149]
- Constrained Optimization by Linear Approximations (COBYLA) [150]
- Nelder-Mead simplex (NELDERMEAD) [151]
- Suplex (SBPLX) [152]
- NEWUOA [153]

To find the optimal algorithm for this application, one of them was run for several minutes with an example Anchor Cloud to find the lowest achievable value for Equation 9.6. Each of the algorithms listed above were then run until they achieved this solver value with an acceptable tolerance ($\pm$ 0.001 of the target solver value). This was repeated 5 times and the mean solver run times are shown in Figure 9.2.

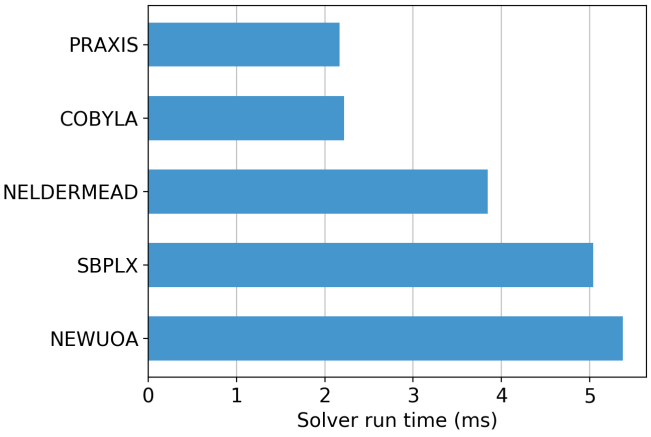


FIGURE 9.2: Comparison of mean solver run times for selected Nlopt algorithms

In these results, the COBYLA and PRAXIS algorithms tied for the fastest algorithm. COBYLA was ultimately chosen because the NLopt documentation suggests that this algorithm is more robust [148]. However, because the solver run times were all on the order of a few milliseconds, any of the listed algorithms would have been suitable.

In addition to selecting an optimal solver algorithm, an important part of non-linear solving is the initial estimate for the input values, in this context the roll, pitch and yaw

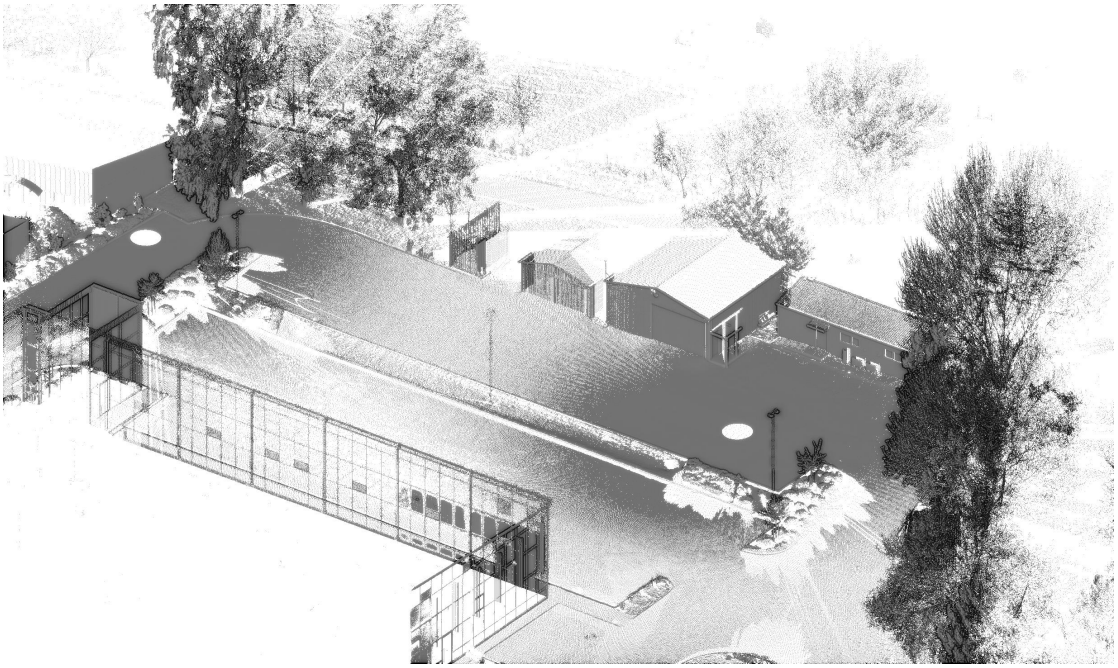
values required to align the paths. A global optimizer will search the entire problem space for the global minimum. But a local optimizer will simply converge to the first minimum that meets the optimizer's stopping criteria, in which case the result is not guaranteed to be the global minimum. Running the solver multiple times with different initial conditions essentially gives it multiple chances to find the global minimum. The results from each run can then be easily compared and the one with the lowest solver error selected.

From observation, any discrepancy between the pre- and post-registration paths occurs as yaw rather than roll or pitch. The magnitude of this error varies but can be as much as 10°. So, five initial estimates were chosen with roll and pitch set to 0°, and yaw values of 0°, -10°, +10°, -25°, or +25°.

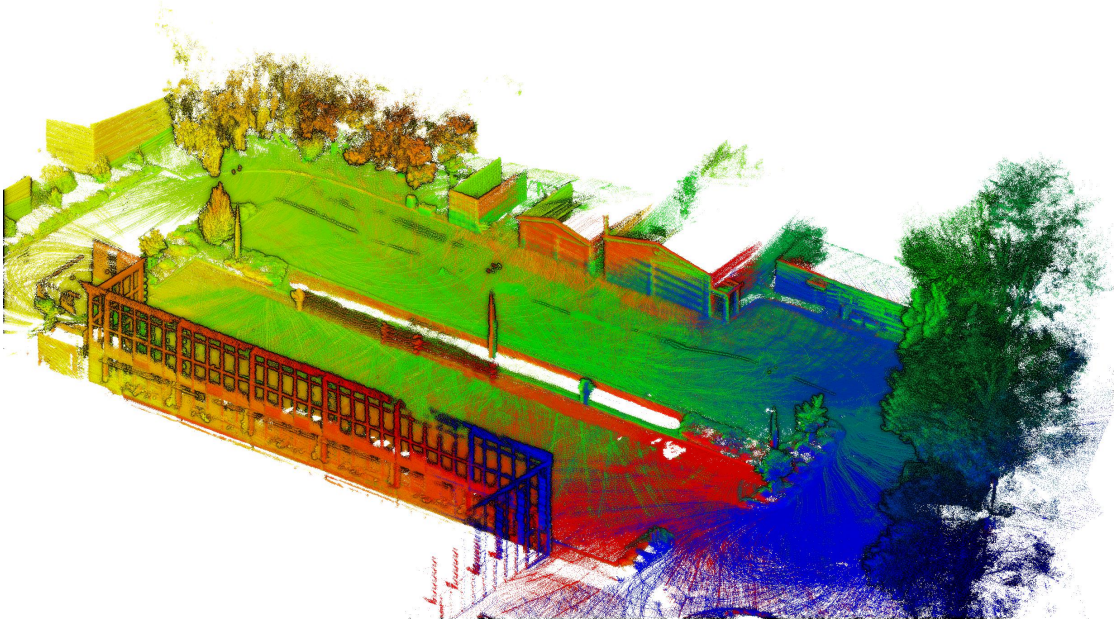
The intuition being that ±10° is the reasonable estimate for the correct re-alignment of the paths (i.e. the global minimum). The 0° and ±25° estimates are then guaranteed to lie either side of correct alignment. If there is a local minimum near the global minimum, the solver will have the opportunity to approach it from both ends.

Once the five solver runs have been completed and the lowest solver error is chosen, the corresponding rotation is applied to the post-registration path, and to the Anchor Cloud. Then various quality control measures are taken to mitigate any residual error in the Anchor Cloud, as discussed in next section.

The end result is (as per the design principles) a set of independent sub-maps (Anchor Clouds) which are earth-referenced in UTM coordinates. Figure 9.3 provides an isometric view of an example ACM output, with a point cloud collected by a Trimble SX10 total station for reference [154]. The complete ACM cloud (i.e. the aggregation of the 13 successfully constructed Anchor Clouds in this data set) has approximately 37 million points (after down sampling) and took less than four minutes to collect. It is worth noting that the ACM cloud provides a much denser and more evenly distributed point cloud than the SX10, even though the ACM cloud is less accurate.



(a) Ground truth SX10 point cloud



(b) ACM point cloud - Each Anchor Cloud is assigned a different colour

FIGURE 9.3: Visual comparison between SX10 ground truth point cloud and ACM result. 13 Anchor Clouds are displayed, each with a different colour.

9.1.5 Quality Control

Limiting error in each Anchor Cloud is important for improving overall accuracy. The methods implemented in this research include:

1. Removing Anchor Clouds with short baselines
2. Removing Anchor Clouds with a large solver error (indicating failed construction)
3. Limiting the maximum radius of registered clouds

Anchor Cloud baseline length

Careful selection of the Anchor Poses is critical, because not only do they require the robot to have stopped and remained stationary, but there need to be enough viable clouds on either side to form a long baseline. For example, the robot is stationary at the beginning and end of its trajectory. But if they were constructed as Anchor Clouds, then their baselines would only be able to proceed in one direction, resulting in an overall length of 7m or less, rather than the ideal maximum length of 14m. So the first and last poses are ignored as potential Anchor Poses.

In addition, the robot may need to stop more frequently than every 7m to facilitate a smooth trajectory through the environment. Enforcing a spacing of no less than 6m between Anchor Poses ensures that every constructed Anchor Cloud will have a corrective baseline of between 12-14m. If two potential Anchor Poses are too close together, the one with the larger number of stationary poses that can be averaged is selected, whilst the other is deleted from the list of potential Anchor Poses.

Anchor Cloud solver error

The solver error for each Anchor Cloud is used to align the pre-and post-registration paths, but it is also an indication of the quality of the Anchor Cloud. The primary cause of an inflated solver error is when one cloud is not correctly registered during Anchor Cloud construction. If this happens with several clouds, it may result in a sub-optimal alignment. The tables of results in Section 9.3 show example solver errors. From observation, a well-constructed and aligned Anchor Cloud as a solver error of less than 1.0. From inspection of the data sets used in this research, Anchor clouds with a solver

error larger than this always contain at least one improperly registered cloud. Therefore, in keeping with the design principle of removing failed sub-maps, any Anchor Cloud with a solver error greater than 1.0 is ignored.

Cloud maximum range

One of the more important variables to control is the maximum range of each LiDAR cloud before it is used to construct an Anchor Cloud. A large maximum range increases the extent of the mesh used to register clouds with MGICP (Chapter 8), which can improve the accuracy of registration. The disadvantage is that because there will always be a non-negligible amount of rotational error, far away points will have positional error that is a function of the rotational error and distance of the point from the LiDAR unit.

For each individual point, the amount of error cause by rotational error in the placement of the whole cloud can be approximated as the chord of a circle (ignoring translation error) as shown in Figure 9.4. Where the length of the chord is the distance between the point's correct ($\mathbf{p}_i$) and actual ($\mathbf{p}'_i$) position. Using the equation for a chord, by setting the maximum tolerable chord length and expected rotation error, the ideal maximum radius r_{max} can be determined.

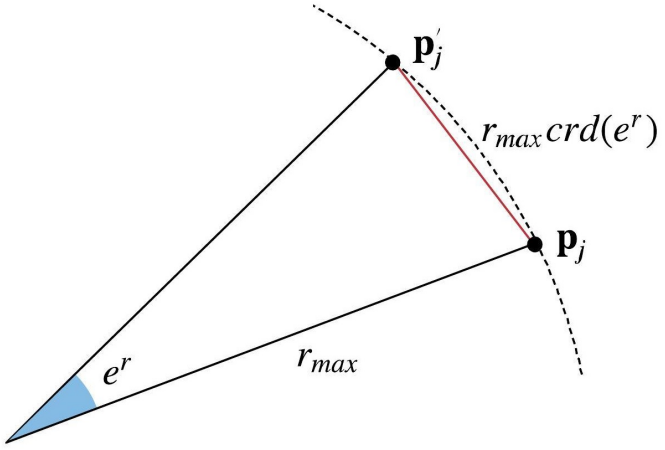


FIGURE 9.4: Chord diagram

Looking ahead to the results of this chapter (Section 9.3), the median rotational error is approximately 0.5°. Adopting the same distance threshold of 0.25m from Chapter 8 and setting it as the maximum desired chord length yields an ideal radius of 28.6m. The practical intuition is then that past this radius, the error in the position of any point is going to have a greater contribution from rotation error than translation error.

9.2 Experimental Setup

9.2.1 Source of Ground-truth

Throughout this research, surveying total stations have been used as the source of ground truth information and this remains true here. Point cloud scans of an office building and adjacent park have been collected using a Trimble SX10 scanning total station. The clouds it produces are fully earth-referenced in UTM coordinates, and each point is accurate to 2.5 mm [154]. Figure 9.5(a) shows an aerial view of the sites used for testing while Figure 9.5(b) provides a top-down view of the ground-truth SX10 point clouds, with the office building and carpark shown in orange, and the park shown in blue.

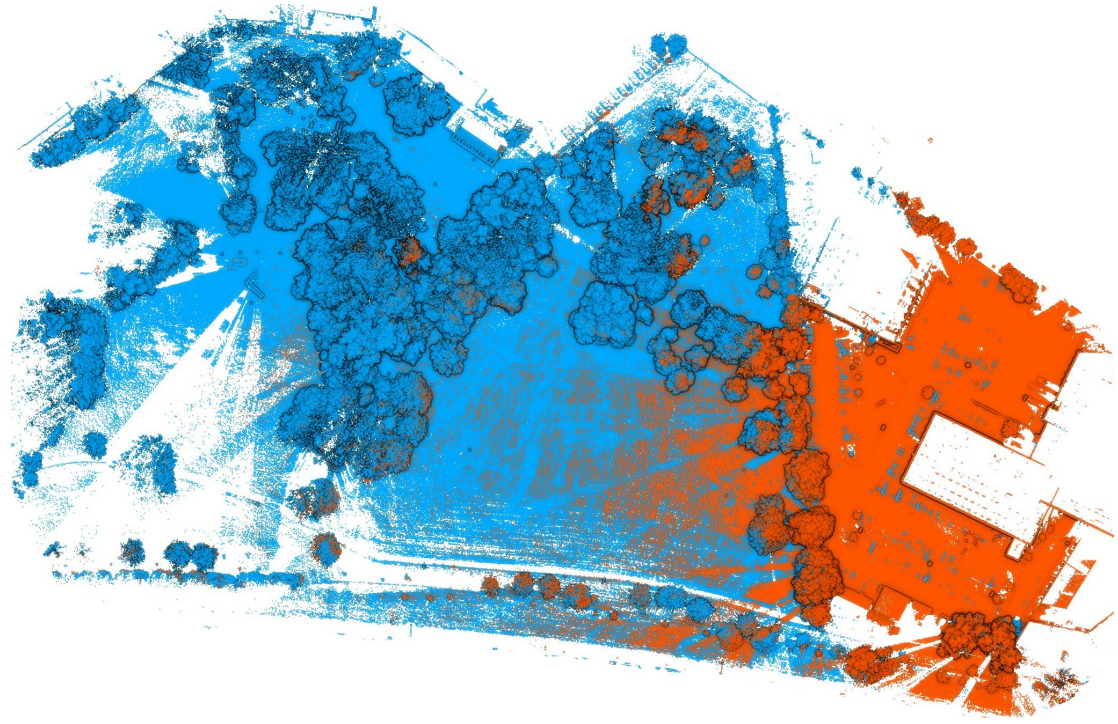
Each total station control point was measured with 3 minute GNSS RTK observations, at different times of day on multiple days. This data was averaged to provide the control points used to calibrate the position of the SX10 and the backsights.

Each SX10 cloud consists of a number of overlapping full dome scans on the "coarse" setting. Each full dome scan generates between approximately 4.2-5.6 million points, and takes approximately 11.5 minutes to complete the scan. Together, the SX10 clouds consist of 11 full dome scans, 4 for the office building and 7 for the park. This yields a total scan time of just over 2 hours. But this time does not include setup, calibration and pack-down, which adds a considerable amount of time to the whole procedure. The park data set was provided by Trimble while the office data set was collected my me.

9.2.2 Data Acquisition

For quantifying the accuracy of the ACM method, extended versions of the data sets used in Chapter 8 were used. These are referred to as the "Garage", "Carpark" and "Forest" data sets. Each data set was collected by having the robotic prototype navigate between a series of waypoints spaced approximately 6 m apart. All data was recorded in a ROS bag file, which was then transferred to a external computer to be run offline through the ACM program. The ACM program loads the bagfile and extracts all sensor and point cloud data before following the methodology outlined in Section 9.1.

(a) Aerial view from Google Maps



(b) SX10 ground-truth point clouds.

FIGURE 9.5: Ground truth data set for ACM testing. The ground-truth office building and carpark are shown in orange, and the park is shown in blue.

During data collection, some data sets were collected with the robot operating autonomously, and others were collected with it being manually driven. For the data sets used here, the Garage and Forest data sets were manually collected, while the Carpark data set was autonomously collected.

9.2.3 Error Metrics

In the case where the end goal is an accurate earth-referenced map, there is no perfect metric for measuring its accuracy. Two metrics can be used: positional/rotational error of each individual anchor cloud, and point-to-point error. Each metric captures different information.

Calculating the translation and rotation error of each anchor cloud is the first metric. This is calculated in the same way as the error metrics used in Chapter 8 (specifically Equations 8.1 to 8.3) for quantifying the accuracy of GICP registration. Each anchor cloud created by ACM has a pose which aligns it with the SX10 ground truth cloud, denoted $\mathbf{P}_i$. Registering the anchor cloud to the SX10 cloud corrects any errors in its position and orientation, and the new pose is denoted $\mathbf{P}_i^{tru}$. The transform from one to the other is then the error transform:

$$\mathbf{E}_i = \begin{bmatrix} \mathbf{R}_i^E & \mathbf{t}_i^E \\ 000 & 1 \end{bmatrix} = (\mathbf{P}_i^{tru})^{-1}\mathbf{P}_i \tag{9.7}$$

And the translation and rotation components of this error matrix are defined in the same way as before. Their equations are provided here again for clarity:

$$e_i^t = |\mathbf{t}_i^{tru} - \mathbf{t}_i| = |\mathbf{t}_i^E| \tag{9.8}$$

$$e_i^r = arccos\left(\frac{trace(\mathbf{R}_i^E) - 1}{2}\right) \tag{9.9}$$

Manufacturers of survey equipment often provide separate horizontal and vertical accuracy values. So the translation error e_i^t can be further split into these components as

follows:

$$\mathbf{t}_i^E = [e_{ix}, e_{iy}, e_{iz}]^T \tag{9.10}$$

$$e_i^{vt} = e_{iz} \tag{9.11}$$

$$e_i^{hz} = \sqrt{e_{ix}^2 + e_{iy}^2} \tag{9.12}$$

These metrics are then the best estimate of the ability of the ACM methodology to accurately place a large point cloud in UTM coordinates.

Calculating the point-to-point error between the ground-truth SX10 cloud and the ACM-produced cloud is the second error metric. This metric encompass all sources of error, including errors introduced by motion-warped point clouds and imperfect registration.

However such a comparison is flawed for two significant reasons. First, the environment contains a great deal of vegetation, which moves in the wind during data collection. Second, the data coverage of the SX10 scans vs the robot are fundamentally different. Not just because their point densities and distributions are different, but also because LiDAR shadows may exist in one data set but not the other. Regardless, the point-to-point error is the best available metric for quantifying the total accuracy of the map, and it can be useful for highlighting where errors exist within each anchor cloud. The point-to-point error is defined as follows:

$$e_i^p = |\mathbf{p}_i^{ACM} - \mathbf{p}_i^{SX10}|, \quad 0 \leq e_i^p < 0.5 \tag{9.13}$$

$$\tilde{e}^p = median(e_i^p) \tag{9.14}$$

$$\bar{e}^p = mean(e_i^p) \tag{9.15}$$

Where e_i^p is the Euclidean distance between point $\mathbf{p}_i^{ACM}$ in the ACM cloud and the closest corresponding point in the ground-truth SX10 cloud $\mathbf{p}_i^{SX10}$. Errors over 0.5 m are discarded, as upon inspection of the data, errors larger than this appear to primarily be the result of LiDAR shadows. However, this only removes the worst outliers, not the errors caused by differences in point distribution and density. The point to point error of the whole cloud $\tilde{e}^p$ is then the median and mean error of it's constituent points. The median error is measured to minimize the effects of outlying points, while the mean is provided because this is the primary error metric used in previous experiments in this

research. So providing the mean as well as median makes comparisons across this thesis more accurate.

To summarize, this section presents five error metrics for defining the accuracy of the ACM method. The pose error for each anchor cloud can be expressed as the rotational error e_i^r in degrees and the translation error e_i^t in millimeters. This can be further decomposed into its horizontal e_i^{hz} and vertical components e_i^{vt}. The last error metric is for the mean $\bar{e}^p$ or median $\tilde{e}^p$ point-to-point error for the whole anchor cloud , also expressed in millimeters.

9.3 Results

The results for each data set are shown in Tables 9.1 to 9.3. They provide the error metrics for each Anchor Cloud in each data set, as well as the median values for each data set.

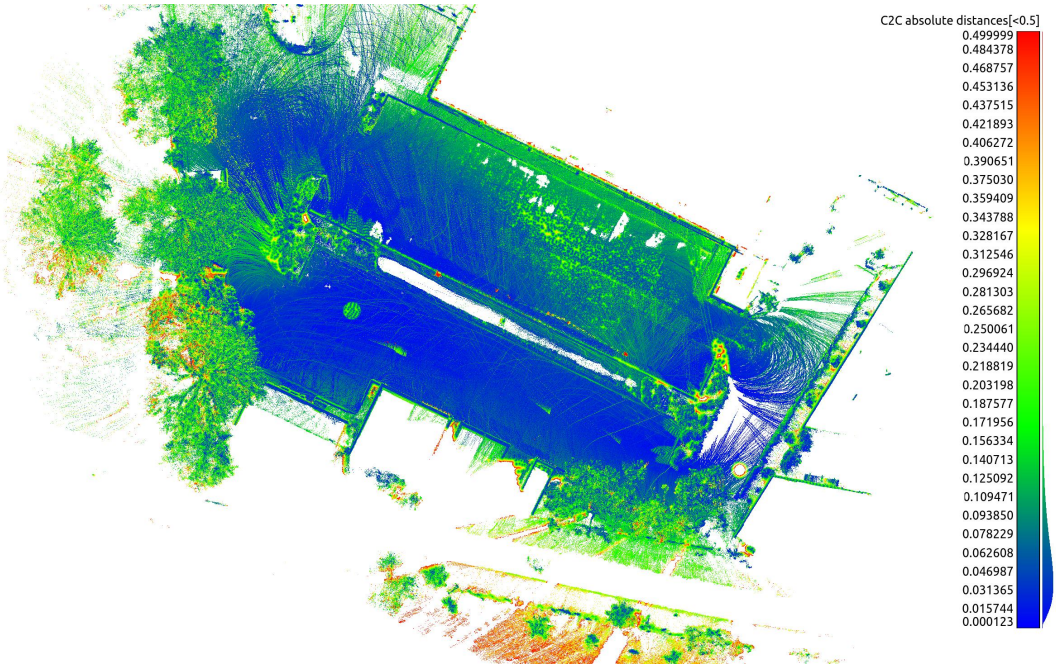
Aerial images of each data set and their real-world locations are shown in Figures 9.6 to 9.8. Recall that every result from the ACM process is automatically placed in earth-referenced coordinates, so no additional adjustments have been made to its orientation. So these aerial images demonstrate that the ACM clouds are correctly positioned and scaled, with no warping which can often occur in SLAM systems.

Figures 9.9 to 9.11 show the same data sets from an isometric perspective, alongside the ground-truth SX10 point clouds. With the ACM coloured by point-to-point error, this view gives a visual indication of what level of error exists in the point cloud and where it is distributed.

9.4 Discussion

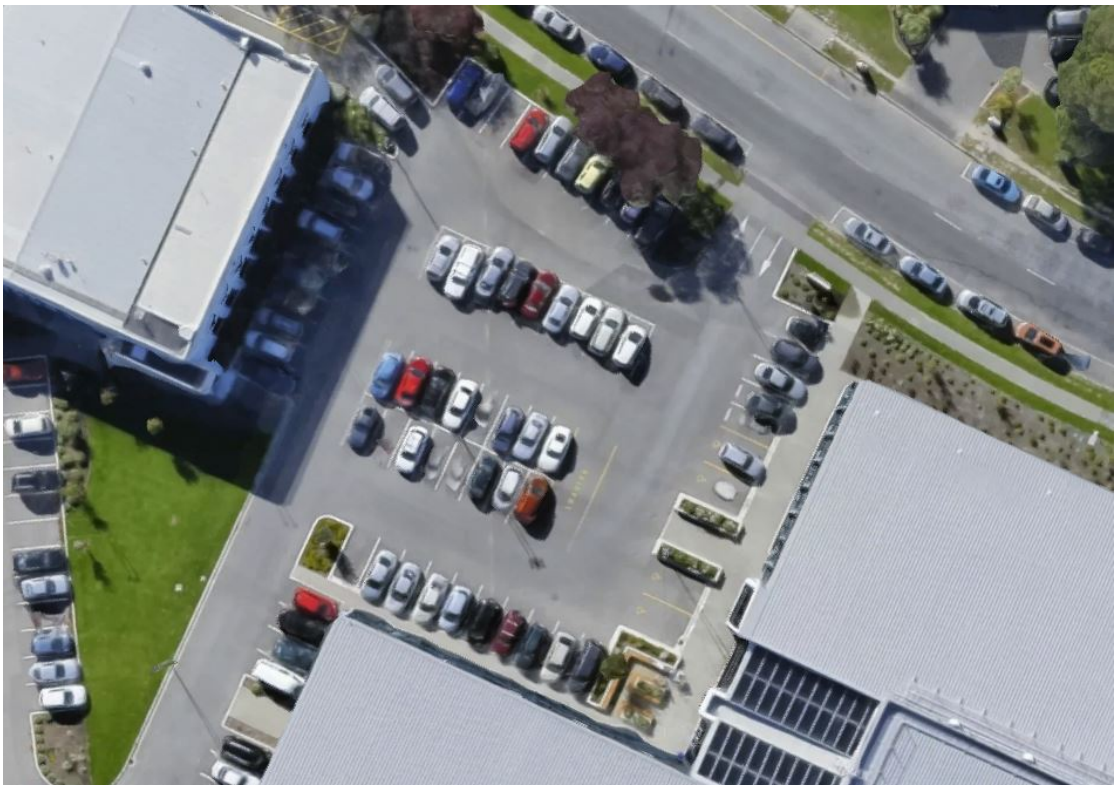
This section will directly address the Research Questions (Section 1.2) by comparing the performance of the robot running ACM to conventional survey methods. Doing so requires making several direct and indirect comparisons between these results and the results of earlier chapters. Some error metrics have been changed and modified to better suit the information they are trying to convey. So a direct comparison is not always possible, but relevant considerations have been noted where necessary.

(a) Real world location

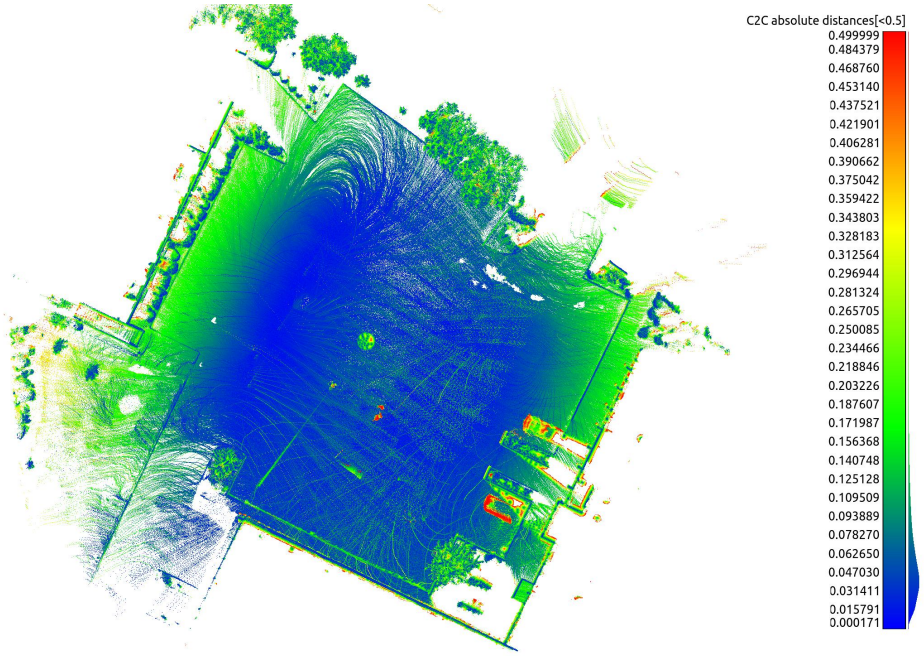


(b) ACM point cloud map. Note that no cars are present in the ACM cloud.

FIGURE 9.6: Aerial view of ACM result for the Garage data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded



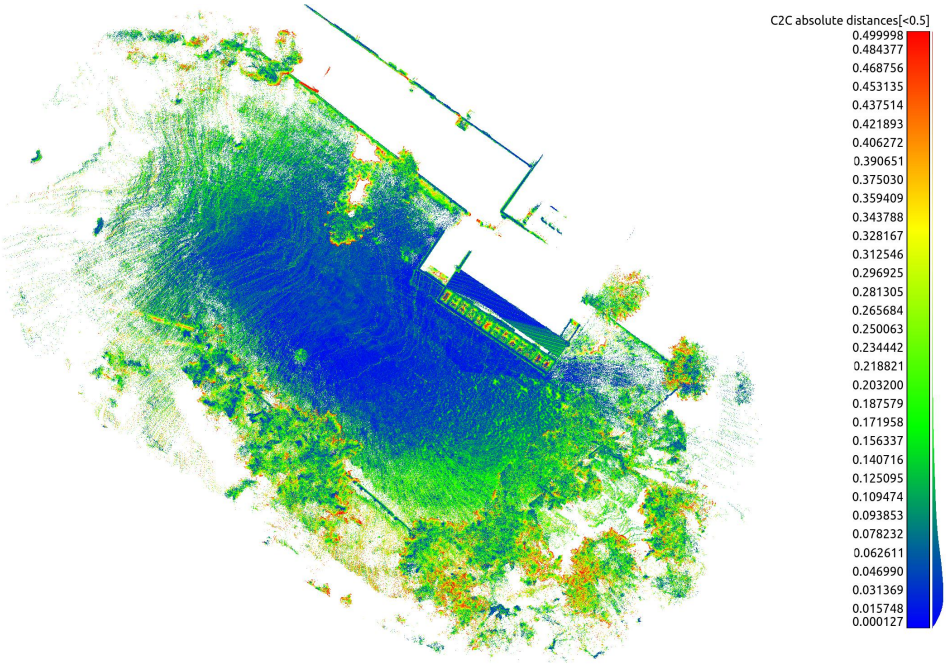
(a) Real world location



(b) ACM point cloud map. Note that no cars are present in the ACM cloud.

FIGURE 9.7: Aerial view of ACM result for the Carpark data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded

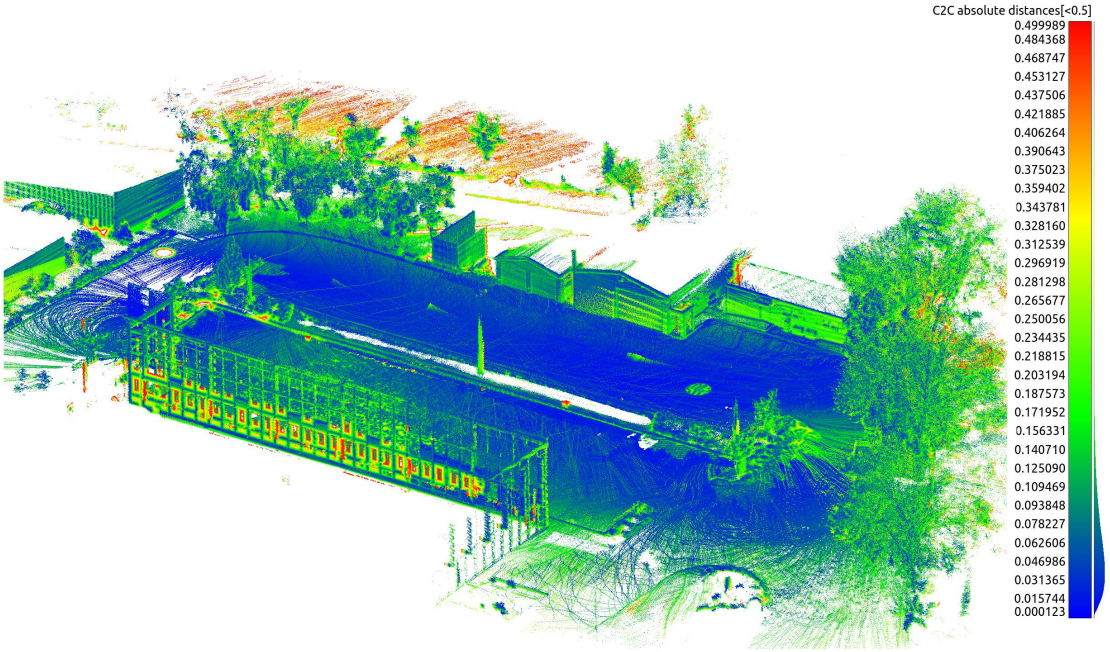
(a) Real world location



(b) ACM point cloud map.

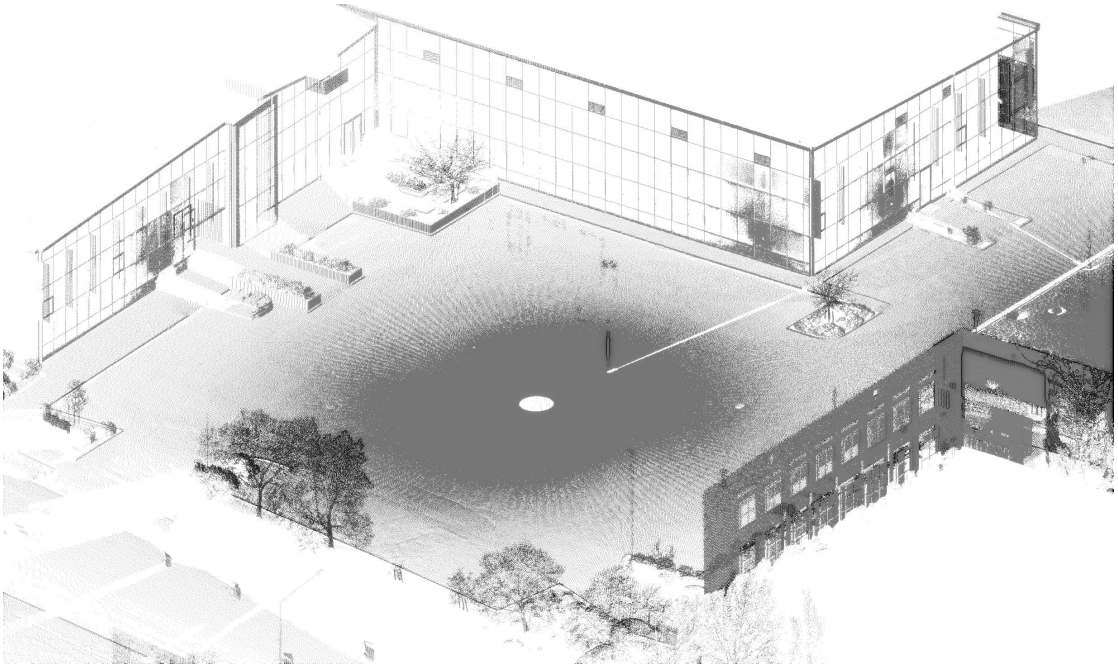
FIGURE 9.8: Aerial view of ACM result for the Forest data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded

(a) Ground-truth SX10 point cloud

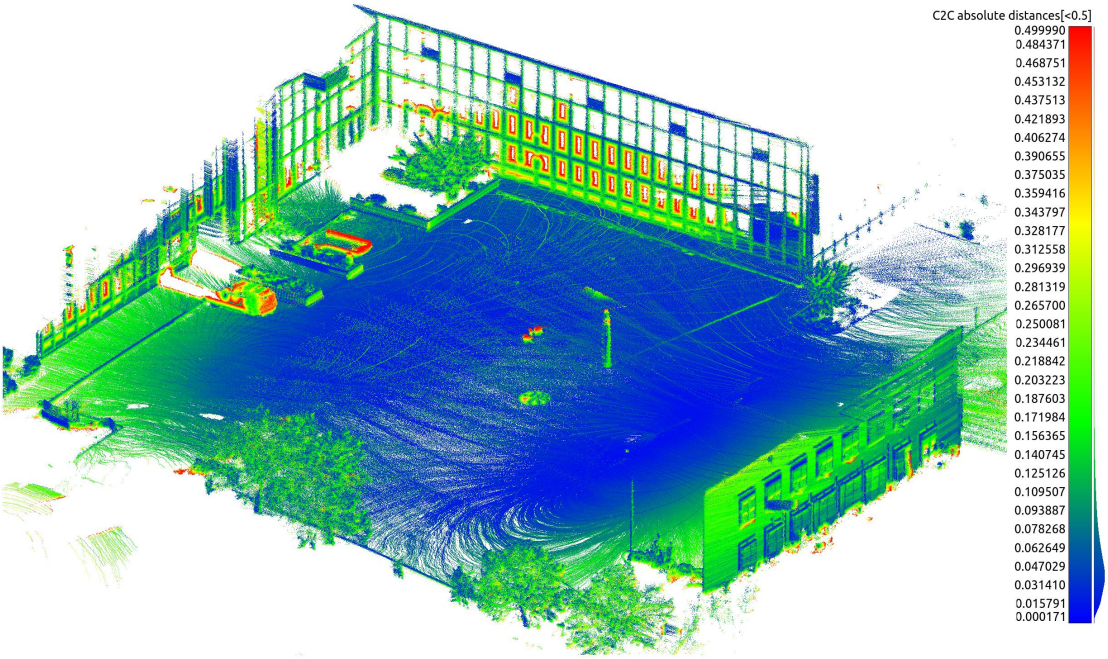


(b) ACM point cloud map.

FIGURE 9.9: Isometric view of ACM result for the Garage data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded



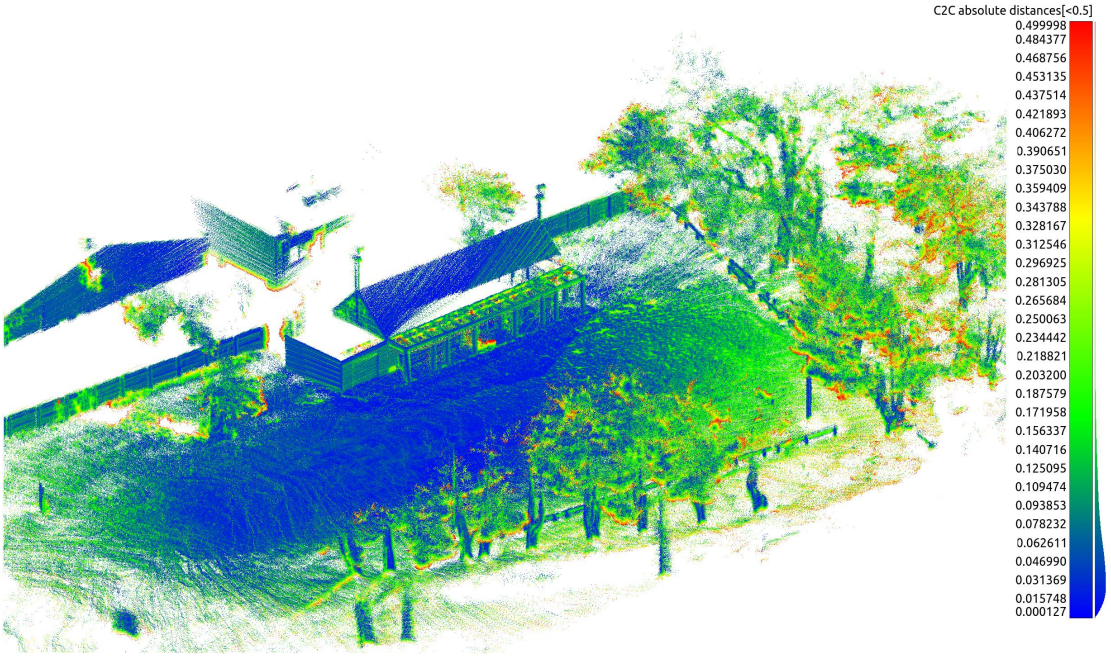
(a) Ground-truth SX10 point cloud



(b) ACM point cloud map.

FIGURE 9.10: isometric view of ACM result for the Carpark data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded

(a) Ground-truth SX10 point cloud



(b) ACM point cloud map.

FIGURE 9.11: Isometric view of ACM result for the Forest data set, coloured by point to point error (in meters). Only points used in the calculation of the results in Table 9.1 are shown, points with errors great than 0.5m are excluded

TABLE 9.1: ACM results for Data set 1: Garage

Anchor Cloud	**Points** ($\times 10^6$)	**Translation Error (mm)** Horz	Vert	Total	**Rotation Error (°)**	**P-P Error (mm)** Mean	Median
1	2.9	55	51	75	0.31	78	57
2	3.4	48	42	64	0.20	89	72
3	3.8	58	55	80	0.21	90	72
4	1.7	40	46	61	0.45	70	53
5	2.7	23	54	59	0.34	67	51
7	3.3	58	46	74	0.36	71	53
8	3.2	40	26	48	0.51	69	40
9	2.3	40	49	64	1.09	83	58
10	4.0	30	60	68	0.47	64	49
11	3.0	10	55	56	0.66	62	51
12	2.5	26	62	68	0.62	65	51
13	2.3	32	59	67	0.61	68	51
14	2.6	52	72	88	0.88	89	68
Mean	2.9	39	52	67	0.52	74	56
Median	2.9	40	54	67	0.47	70	53

Anchor Clouds 6 and 15 are excluded as their solver errors exceed 1.0 (3.06 and 63.80 respectively). Note: With the exception of Translation (vert) error, all values shown are absolute values.

TABLE 9.2: ACM results for Data set 2: Carpark

Anchor Cloud	**Points** ($\times 10^6$)	**Translation Error (mm)** Horz	Vert	Total	**Rotation Error (°)**	**P-P Error (mm)** Mean	Median
1	6.1	74	50	89	0.73	86	51
2	3.2	77	50	92	0.44	80	51
3	2.5	56	66	87	0.14	93	67
7	2.2	55	45	71	0.17	73	56
8	2.1	53	35	64	0.17	66	47
9	2.4	52	51	72	0.16	77	58
11	4.2	41	50	64	0.76	69	49
Mean	3.2	58	50	77	0.48	78	54
Median	2.5	55	50	72	0.50	77	51

Anchor Clouds 4-6 and 10 are excluded as their solver errors exceed 1.0 (39.23, 71.06, 5.39, 2.24 respectively). Note: With the exception of Translation (vert) error, all values shown are absolute values.

9.4.1 Accurate Positioning of Earth Referenced Point Clouds

What is immediately apparent is that as the robotic prototype has become more complex, and more software components are added, the larger the errors in the results become. Section 5.3 shows that the robotic prototype has a base accuracy or 10-20 mm, with a further 10-20 mm worth of error potentially coming from environmental factors.

TABLE 9.3: ACM results for Data set 3: Forest

Anchor Cloud	**Points** ($\times 10^6$)	**Translation Error (mm)** Horz	Vert	Total	**Rotation Error (°)**	**P-P Error (mm)** Mean	Median
1	5.0	21	90	92	0.79	110	87
2	2.8	30	39	49	0.29	64	38
3	2.1	31	4	31	0.31	51	29
4	1.8	22	71	75	0.47	82	62
5	1.9	6	40	41	0.75	85	59
Mean	2.7	22	49	58	0.52	78	55
Median	2.1	22	40	49	0.47	82	59

Anchor Cloud 6 is excluded as its solver errors exceeds 1.0 (2.71)
Note: With the exception of Translation (vert) error, all values shown are absolute values.

Chapter 7 shows how unintended rounding in crucial mathematics can contribute a significant amount of error to the final result. So considering that much of the ACM infrastructure is experimental or open-source, it's possible that a significant amount of the error in the final position of each Anchor Cloud is caused by similar computation errors, rather than an inherent flaw in the ACM methodology. A significant percentage of the remaining error is likely due to the inherent error of the Velodyne LiDAR unit ($\pm$ 20 mm [155]) and the small amount of position and rotation error in each Anchor Cloud.

However, the results of the ACM method can still be compared to conventional survey methods in order to answer the research questions. The vertical translation error of each Anchor Cloud represents the ability of the system to position itself. This data can be compared to the vertical accuracy of the conventional survey methods tested in Chapter 6. For clarity, the relevant values have been repeated in Table 9.4.

Unfortunately there are not enough features in Zone 1/2 for the ACM system to work well (see Section 9.4.5). So by necessity, this comparison has to be made between data sets, which use different methods of calculating the vertical accuracy. Specifically that the data collected in Zone 1 compared approximately 200 points to a ground-truth surface, while the ACM results compared two point clouds, each containing millions of points.

All of these differences and sources of error make the comparison presented in Table 9.4 highly flawed, but it still gives an approximation of how accurate the robotic system is as a whole, when compared to more conventional survey methods.

Although not the most accurate method, the robotic prototype with ACM can achieve a level of accuracy comparable survey-grade instruments and methods. With the added

TABLE 9.4: Mean vertical position accuracy across all survey systems

	Data set				
Method	Zone 1	Zone 2	Garage	Carpark	Forest
TS-Stop[1]	0	0			
TS-Cont	37	10			
TS-Scan (grass)[2]	70	67			
TS-Scan (no grass)[3]	0	2			
GNSS-Stop	6	15			
GNSS-Cont	54	31			
UAV-P	1	3			
Robot (UKF)	8	17			
Robot (ACM)			52	50	49

All errors are given in millimeters. The total station used to collect TS data was a Trimble VX, not a Trimble SX10 total station.
[1] The TS-Stop data was used as the ground-truth data for all other data in this table except Robot (ACM)
[2] Raw, as-measured data without compensation for grass height
[3] 70 mm offset subtracted from all elevation measurements to account for grass height

benefit of being able to operate autonomously, much faster than most of these survey methods, and producing a much higher density

9.4.2 ACM Mapping Accuracy

Comparing the ACM point-to-point error with the SX10 helps to answer the Research Question. Table 9.5 shows the error of their respective point clouds, as well as the time taken to collect each cloud. The SX10 values are taken from its datasheet [154]. A single full-dome scan takes 12 minutes to complete, and generates approximately 4.5 million points. It took roughly two full dome scans to generate the points for each data set, resulting in a total scan time of 24 minutes for approximately 9 million points.

TABLE 9.5: Mapping accuracy of robot/ACM vs SX10 total station

	SX10			**Robot/ACM**			
Data set	Error (mm)	Points ($\times 10^6$)	Time (min)	Mean P-P Error (mm)	Median P-P Error (mm)	Points ($\times 10^6$)	Time (min)
Garage	2.5	9	24	74	53	37.9	4
Carpark	2.5	9	24	78	51	22.7	5
Forest	2.5	9	24	78	59	13.6	2

All times strictly include scanning time only. They do not include setup, pack-up or processing time. SX10 error and time are copied from the datasheet, assuming at least two full dome scans are required per data set.

LiDAR sensor error, warping due to motion, error in the position or orientation of the Anchor Clouds, wind in the environment are all factors that increase the measured point-to-point error. So it should be expected to be larger than the translation error.

It should also be noted that while the SX10 takes only 12 minutes to complete a full dome scan, it can require an hour or more to set up each control point, perform back site measurements and shift total station positions. Hence while Table 9.5 shows only scan times, in reality, collecting all the SX10 point clouds took approximately 3 days of work to complete. Whereas the robot required less than 10 minutes of setup before collecting its data. If operated autonomously, then the time taken for the robot to map an area is inconsequential to the user, and leaves them free to perform other tasks

Considering all possible sources of error, a median error of approximately 50-60 mm is an excellent result when compared to survey-grade instruments, and is sufficient for basic terrain mapping. For this purpose, the robot can also collect far more data, collecting up to four times the number of points. Inspection of Figures 9.9 to 9.11 also show that the points are more evenly distributed throughout the environment. This is because the SX10 scan points are separated by degrees, not horizontal or vertical coordinates. So the density of any region of the point cloud is inversely proportional to its distance from the SX10. By comparison, the robot moves as it scans, better distributing points throughout the environment and reducing LiDAR shadows.

The overall results of the ACM system also show that the policy of keeping Anchor Clouds separate, and removing ones with a solver error greater than 1.0 is successful in limiting errors when they occur. Using the Carpark data set as an example, several Anchor Clouds were constructed with misaligned clouds (as evident from their large solver errors). In a typical SLAM system, these errors might have compromised the integrity of the map from that point onwards. Even with 6 Anchor clouds removed from that data set, there were still enough points to adequately survey the carpark.

9.4.3 Environmental vs. Robotic Error

Determining what percentage of error was caused by the robot, and what was caused by the environment is difficult in this situation because there are so many sources of error present at once. Some information can be gleaned from existing work that isolates and measures individual sources of error. Known sources of error on the robot consist of the following:

1. LiDAR range error (±20 mm as per the HDL-32 datasheet [155]).
2. GNSS error in ideal conditions (±8 mm horizontal and ±15 mm vertical error as per R7 datasheet [4])
3. Error caused by motion of the platform. The results of Chapter 6 show that a moving robotic platform will have a vertical RMSE of approximately 14 mm in an ideal environment. Note that this figure inherently includes GNSS error.
4. Point-to-point error caused by error in the orientation of the robot. The maximum radius of the point cloud was restricted to mitigate this error, as discussed in Section 9.1.5. However as Figure 9.14 will later show, up to 250 mm of error may still be present at the outer reaches of the ACM cloud.

Rounding error in point cloud processing, as covered by Chapter 7 should be mitigated to negligible levels by taking the steps described at the end of the chapter. When it comes to environmental error, there are also many sources that have been isolated by this research or other works. The sources most relevant to the results in this chapter are:

1. Vegetation. Section 6.4.2 discusses how grass added approximately 55 mm of error to the TS-Scan datasets, specific to the height of the grass at the time. This source of error will be present in all three ACM datasets, but particularly in the Forest dataset.
2. GNSS-obstructing terrain such as trees or buildings. The GNSS-Stop results of Chapter 6 show an increase of approximately 15 mm due to nearby trees. Other research shows the errors can be much larger in such situations, up to 1 m or more [40, 51, 52].

3. Multipath. This will contribute to error in the Carpark and Garage datasets. The range of elevations reported in the 1a dataset collected during GNSS calibration in Section 5.3 suggests that multipath could have contributed approximately 10-20 mm of error.

4. LiDAR shadows. These will cause false errors in the comparison simply because some parts of the environment were not scanned by both the robot and the SX10.

Given the variety of sources and magnitude of these errors, it is difficult to say with certainty what errors sources predominantly contributed to the results in Tables 9.1 to 9.3. However, visual inspection of Figures 9.9(b), 9.10(b), and 9.11(b) shows that the error increases (becoming green to red) in the vegetation and around the edges of the scan, whilst it is most accurate nearest the trajectory of the robot. This suggests that vegetation and roll error in the ACM are the biggest sources of error. It should be emphasized that this is a hypothesis, and that a comprehensive analysis of these errors (while outside the scope of this research) would make for valuable future work.

9.4.4 Solver Error Correlation

The solver error is crucial for determining which Anchor Clouds have been properly constructed. If this error were correlated with the other error metrics, it would be valuable information that could be used to make further generalizations about the quality of the complete Anchor Cloud.

However, as only a few constituent clouds may be mis-aligned, they often do not contain a large enough percentage of the total points in the Anchor Cloud to negatively effect calculation of the error metrics described in Section 9.2.3. Figure 9.12 shows a correlogram of the primary Anchor Cloud error statistics given in Tables 9.1 to 9.3. The off-diagonal scatter plots include a fitted regression line and 95% confidence interval, while the diagonal plots show the data distribution.

This correlogram shows that there is not strong correlation between the solver error and the other key error metrics. This is further confirmed by the associated R^2 correlation coefficients, all of which are low and close to zero. The largest R^2 value, for translation error vs median point-point error, is 0.419. This indicates that over half the variation in

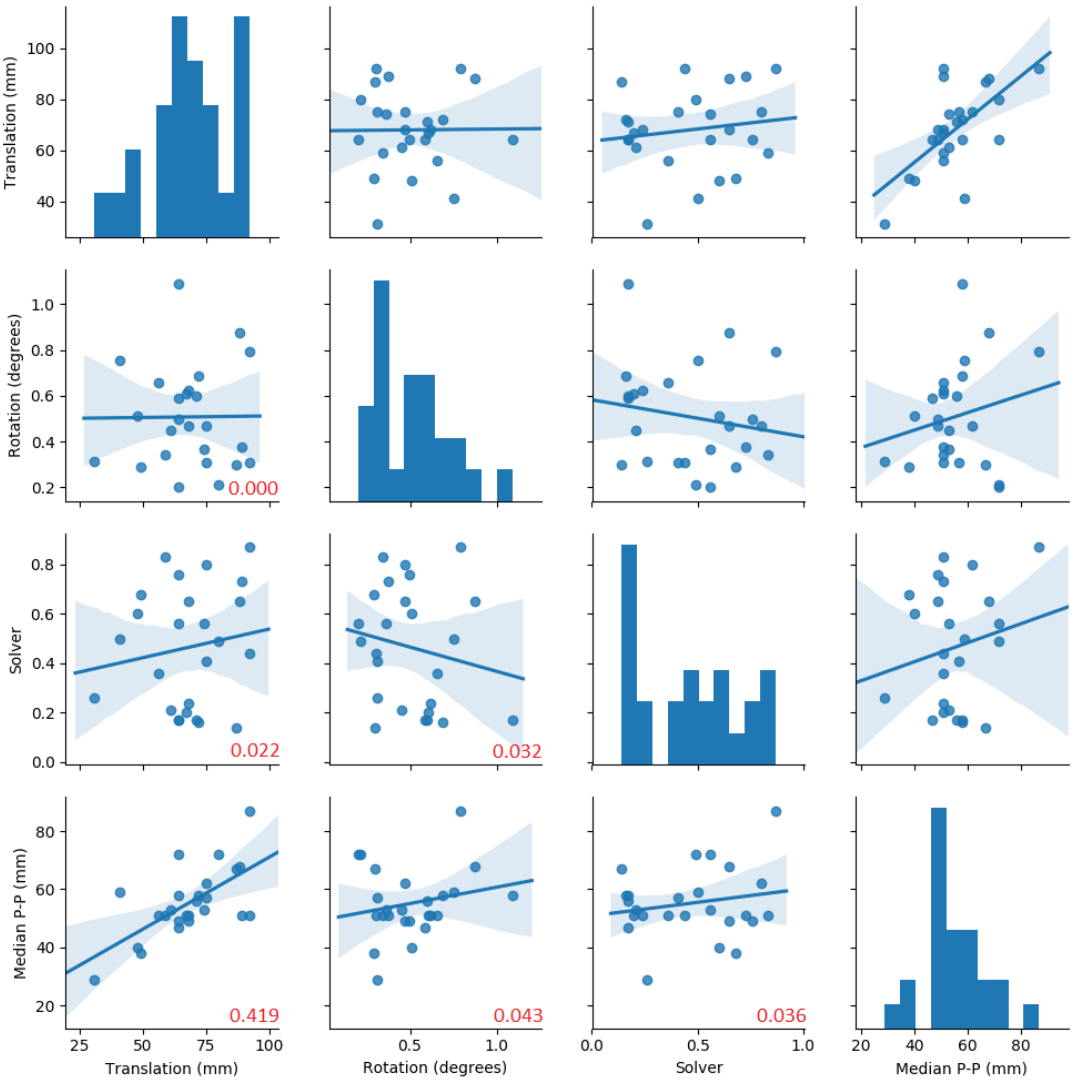


FIGURE 9.12: Correlogram of Anchor Cloud error metrics, including regression line and 95% confidence interval. The R^2 coefficient of determination is shown in the bottom right corner of each plot in red.

the median error is not represented by the regression model, i.e. that they are not highly correlated.

This means that any Anchor Cloud with a solver error of 1.0 or less can be considered accurate enough. The solver error value itself cannot be considered an indication of the specific accuracy of the cloud itself.

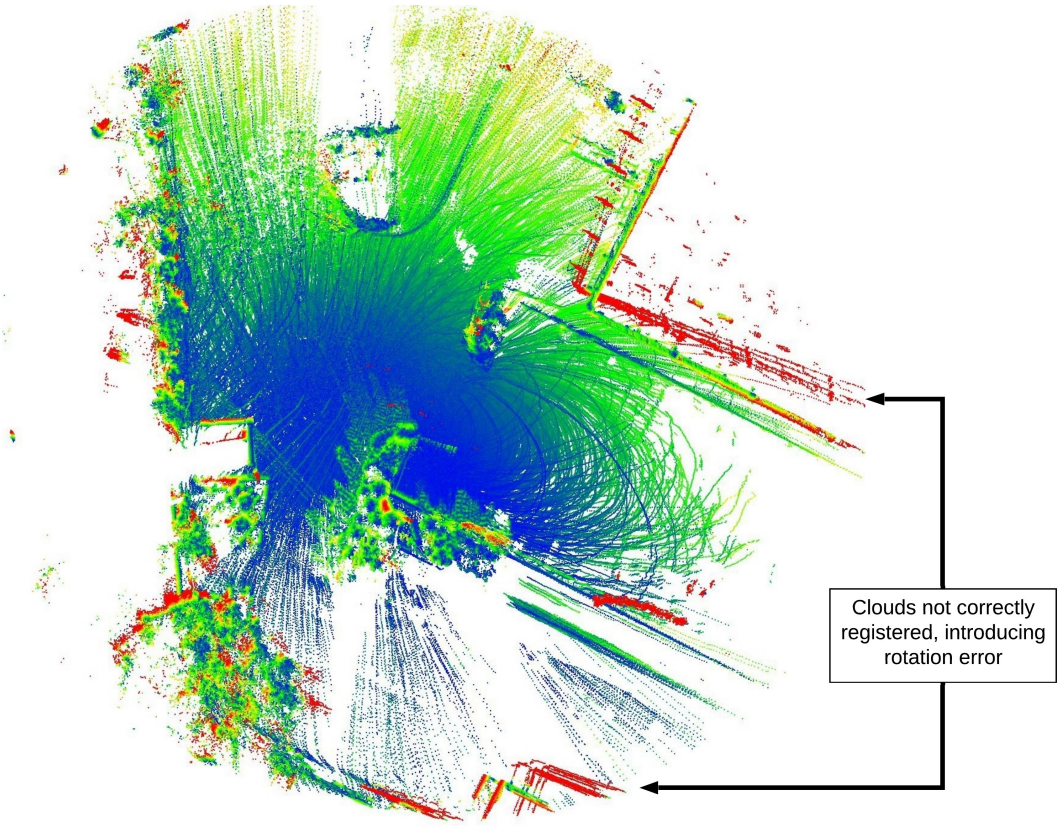
9.4.5 ACM Limitations

There are several limitations of the ACM method. The first is that accurate registration, even with MGICP still relies on there being significant structures in the environment, such as walls or buildings. If too much of the environment is organic and unstructured, the underlying mesh is too varied in nature. This results in Anchor Clouds where many of the constituent scans have been misaligned, and the path alignment therefore fails. Figure 9.13(a) shows an example where Anchor Cloud 7 from data set 1 was created with 2-3 misaligned clouds which appear as bright red points. In contrast, the next Anchor Cloud in sequence (Figure 9.13(b)) was created correctly.

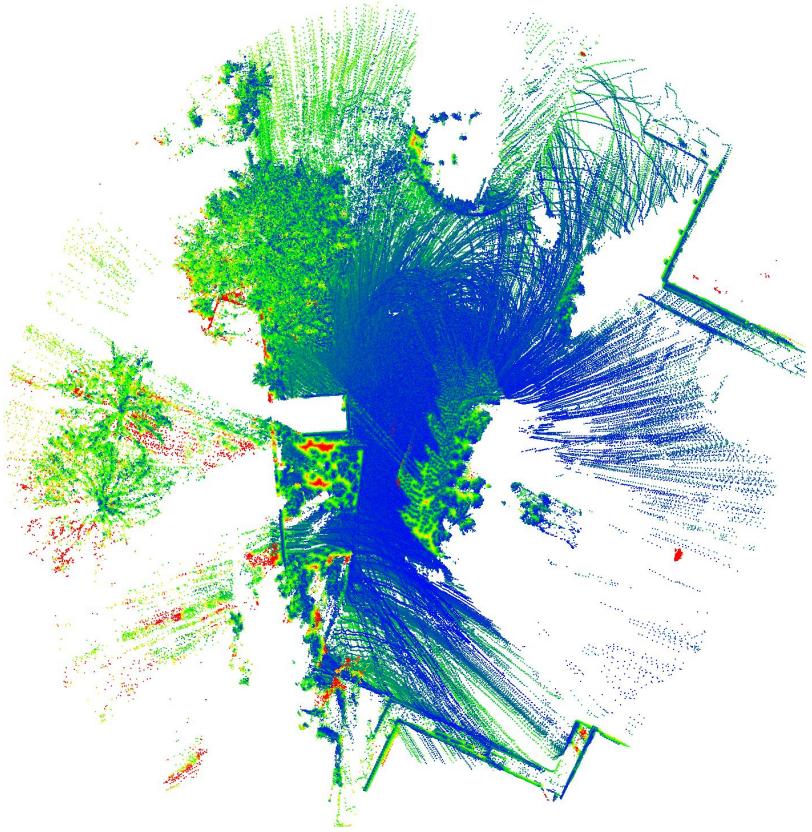
This means that while ACM mostly works well around urban and semi-urban environments, conventional tripod-based surveying or aerial photogrammetry may still be preferable in rural environments.

Another limitation is that the path alignment process is still reliant on the roll of the original UKF poses. Unless a potential Anchor Pose lies in a curved part of the trajectory, the roll is likely to still contain a non-zero amount of error. This is why, when inspecting an individual Anchor Cloud created in a flat area, the point to point error appears greater the further the points are from the path of the robot. Figure 9.14 shows an example anchor cloud where the *vertical* component of the point to point error increases as a function of the perpendicular distance from the robots trajectory. The Anchor Cloud has rolled such that its starboard side is lower than the true topography while the port side is higher.

This is one of the primary motivations for limiting the maximum radius of the complete Anchor Clouds, as discussed in Section 9.1.5. The histogram on the right hand side of Figure 9.14 shows that while most points are clustered with a mean around 0 mm error, there is a distribution of points which have an error of -250 mm or higher.



(a) Anchor Cloud 6 (failed with solver error of 3.06)



(b) Anchor Cloud 7 (succeeded with solver error of 0.56)

FIGURE 9.13: Example of successful vs failed Anchor Cloud from data set 1. All points, even those with a point-point error greater than 0.5 are shown.

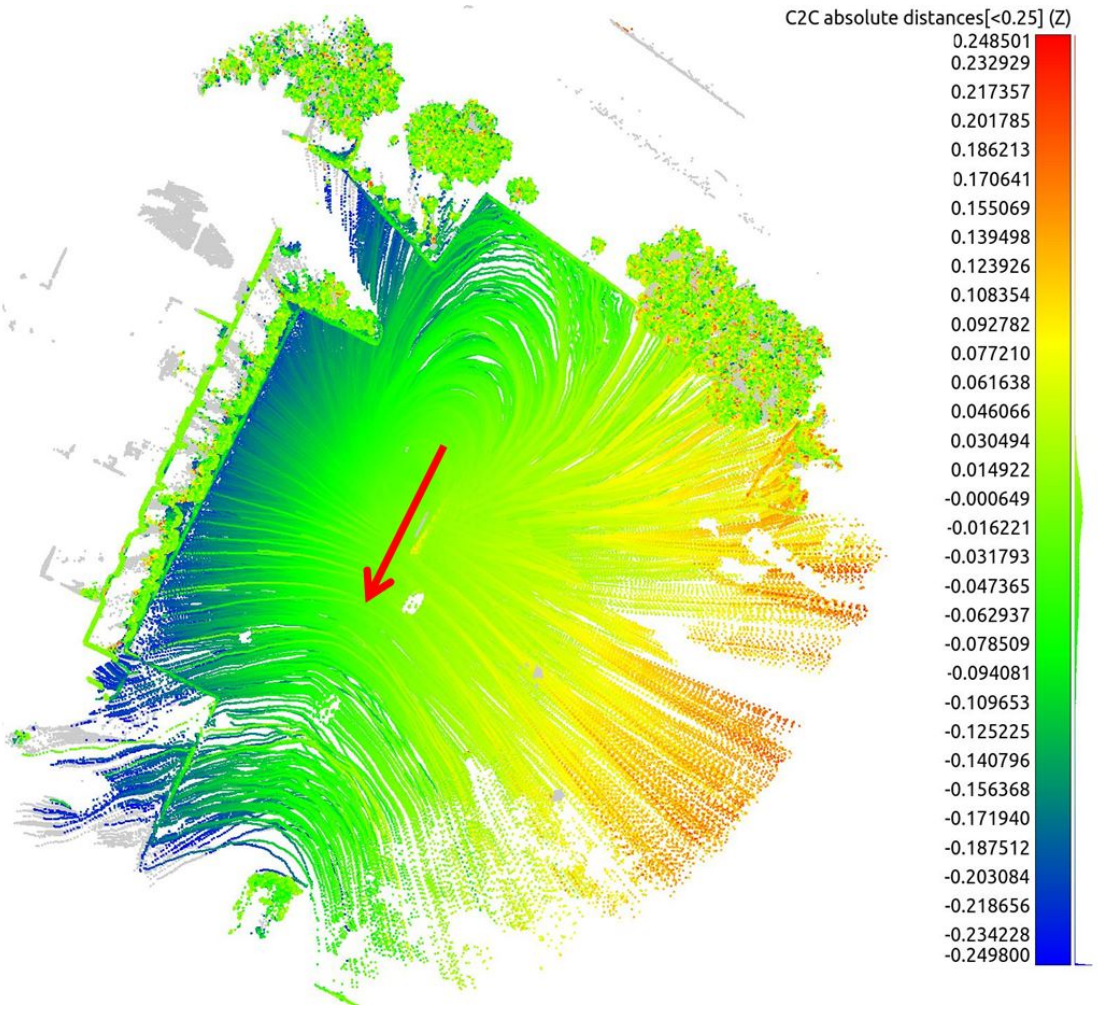


FIGURE 9.14: Anchor Cloud 7 from data set 2 demonstrating roll error. The red arrow denotes the direction and length of the robots trajectory. Points are coloured by vertical error. Note that red is positive while blue is negative.

9.4.6 Summary of the ACM method

Throughout this thesis, several different methods have been used to accurately position a surveying robot:

- Chapter 5 introduced the robotic platform and showed that when using GNSS data alone to statically measure a control point it could achieve a mean vertical accuracy of approximately ±10 mm (excluding outliers). This chapter also showed changes in satellite constellation and environmental factors could add 10-20 mm of error to elevation measurements.
- Chapter 6 then used the robot to survey two test sites, using data from an Unscented Kalman Filter (UKF). This achieved a mean vertical accuracy of 8-17 mm

depending on the data set. Alongside the results of the previous chapter, this suggests that the "base" accuracy of the robot is approximately 10-20 mm.

- Chapter 8 conducted several experiments with cloud registration. This showed a mapping-style method of registration using keyscan Mesh Generalized Iterative Closest Point (MGICP). This method could position a point cloud with an error of 250 mm or less, and a rotation error of 1.5° or less up to a distance of approximately 7 m.

- Chapter 9 combines the research and lessons learned throughout the thesis into the novel ACM method, which can position large, aggregated point clouds with a median translation error of 49-72 mm and median rotation error 0.17-0.47°. The median point-to-point error of the ACM method when compared to a ground-truth cloud is 51-59 mm.

The ACM method itself takes inspiration from how total stations and GNSS systems are calibrated. It first exploits pauses in the robots trajectory to collect and average sequential position measurements, thereby calibrating the position of the robot at that point (called an Anchor Pose). It then registers every cloud within a certain distance of the Anchor Pose *to* the cloud at the Anchor Pose. The aggregation of these registered clouds is the Anchor Cloud, and it is then fixed to the Anchor Pose and it's orientation is calibrated by using a non-linear solver to align it's pre- and post-registration poses. Anchor Clouds with high solver errors at the end of this process are removed.

When compared to the SX10, the robot running ACM is orders of magnitude faster at surveying an environment, taking minutes instead of hours. And it can do this autonomously, leaving the user free to engage in other tasks. It can collect several times more million points (assuming a the SX10 is scanning in "coarse" mode) with a more even distribution. It is less accurate than the SX10, but can still deliver a point cloud that is accurate enough for terrain mapping.

Importantly, ACM also overcomes the flaws in SLAM-generated maps by explicitly using the GNSS information in a UKF to generate an earth-referenced trajectory for the robot. This is the foundation for initial placement of every point cloud, and the local point cloud maps (Anchor Clouds) that are created later. By keeping each Anchor Cloud independent, ACM avoids errors in one being propagated to the next. It also avoids the

entire map being distorted. Removing failed Anchor Clouds removes misaligned clouds that could compromise the accuracy of the overall result.

This features, and the results shown in Tables 9.1 to 9.3 prove that the robot developed for this research, and the ACM method, is a successful solution for automating terrain mapping.

10 | Conclusion

10.1 Thesis Summary

The motivation for this thesis was to investigate how mobile robotics might automate terrain mapping for the geospatial surveying industry. Specifically, to answer the following research questions:

"How does a survey conducted by an autonomous mobile robot compare to a survey conducted by conventional methods?"

and

"How accurate is a point cloud map created by an autonomous mobile robot compared to a map generated by a conventional scanning survey method?"

To answer the first question a mobile robotic platform was developed using off-the-shelf sensors and open-source software. When equipped with a variety of sensors and an Unscented Kalman Filter (UKF), it was used to autonomously conduct a basic elevation survey of two test sites, along with a selection of conventional survey methods. The robot performed with a mean error of approximately 8-17 mm, compared to the mean errors of the conventional methods which fell in the range of 0-70 mm. In terms of survey time, the robot took approximately 13-14 minutes while the conventional methods took either a few minutes or approximately an hour.

To answer the second research question, the robotic platform was further upgraded with a LiDAR unit. A novel method for earth-referenced mapping was then devised which took inspiration from how survey instruments are calibrated. Termed *Anchor Cloud Mapping* (ACM), it registers clouds into independent sub-maps, fixes them to calibrated stationary points, and then corrected errors in orientation by aligning the pre- and post-registration trajectories. When compared to a Trimble SX10 scanning total station, this system could achieve a median point-to-point accuracy of 51-59 mm, with several times more data in a time on the order of minutes. By comparison the SX10 is accurate to 2.5 mm, but in total will take several hours to map the same environment.

This research proves that terrain mapping can be successfully automated by an autonomous mobile robot. While the robotic prototype combined with ACM is not more accurate than conventional survey methods, it is comparable. And it delivers the same point cloud output that a tripod-based scanner does, but is denser, covers more of the environment and has the benefit of it being achieved almost completely autonomously. These are benefits inherent to the design of the robot vs the more static nature of conventional survey instruments. The largest downside is the lower accuracy.

In developing the robot and ACM, several novel contributions to the field of robotics and geospatial mapping have been made. They are:

1. A comparison of a mapping robot with conventional survey methods.
2. An analysis of Loss of Significance (LoS) in cloud registration.
3. A novel method for accurate earth-referenced LiDAR mapping with an autonomous mobile robot.

The robotic prototype and ACM have been constructed largely from hardware and software that is not at a survey grade in terms of accuracy and precision. This is made particularly clear by problems such as LoS which have occurred in the Point Cloud Library (PCL). But this also means that there is significant room for improvement, and that a thorough analysis of the software could yield some critical changes to algorithms and parameters which dramatically reduces the error of the output. It is highly likely that other problems similar to LoS exist, and have already been solved in the commercial software that runs most survey instruments. So there is strong motivation to continue this work and further improve ACM and the robot prototype.

In particular, the methodology of ACM overcomes some of the inherent shortcomings of SLAM, and delivers a point cloud that is still accurate and relatively complete, even where errors occurred during the construction process. This kind of explicit fault detection is necessary for robotic systems to achieve the level of reliability that is demanded in the geospatial industry. Particularly when faults in other systems (SLAM or otherwise) can often compromise the integrity of the whole result. This is why dedicated, built-for-purpose systems like ACM need to be developed for specific applications like terrain mapping.

10.2 Future Work

The bulk of the work in this thesis has been dedicated to achieving a particular contribution - a novel method for accurate earth-referenced mapping. As already stated, there is plenty of future work that could be conducted with the aim of analyzing existing sources of error in the robotic prototype or ACM, and further reducing them. Some of the following ideas have been considered as means of achieving this:

- Investigate how different accelerations and top speeds for the robot may affect warping of LiDAR clouds and therefore the accuracy of the completed Anchor Cloud.
- Expand the non-linear error function for path alignment to account for the linearity of the path. E.g. if it has high curvature, de-weight the orientation as the paths are better aligned by position.
- Complete an error analysis of the ACM system so that sources of error in the Anchor cloud position and orientation can be further reduced. This would search for any other rounding or numeric errors in the system such as LoS.
- Re-survey the same areas with an SX10 to quantify the expected difference between point clouds due to changes in vegetation, wind etc. This would account for some of the error in the ACM results.
- A break down of each Anchor cloud into it's constituent clouds, so their errors can be quantified on a cloud-by-cloud basis, allowing a more accurate comparison with the results of Chapter 8.
- Investigate expanding the ACM concept with a pose graph for globally optimizing the location of overlapping anchor clouds whilst still excluding poorly constructed clouds.

11 | Bibliography

[1] U.S. Army Corps of Engineers. Control and topographic surveying. Manual, U.S. Army Corps of Engineers, 2007.

[2] Trimble Inc. Trimble R10 GNSS system datasheet. Report, Trimble Inc, 2014. URL http://trl.trimble.com/docushare/dsweb/Get/Document-625158/022543-544E_TrimbleR10_DS_1014_LR.pdf.

[3] Trimble Inc. Trimble VX Spatial Station Datasheet. Report, Trimble Inc, 2013.

[4] Trimble Inc. Trimble R7 GNSS system datasheet. Report, Trimble Inc, 2007. URL http://trl.trimble.com/docushare/dsweb/Get/Document-365914/022543-367D_R7GNSS_DS_0213_LR.pdf.

[5] Trimble Inc. Trimble UX5 HP Unmanned Aircraft System. Report, Trimble Inc, 2016.

[6] P. R. Wolf and C. D. Ghilani. *Elementary Surveying: An Introduction to Geomantics*. Pearson Prentice Hall, United States of America, 11th edition, 2006. ISBN 0-13-148189-4.

[7] J. M. Anderson and E. M. Mikhail. *Surveying: Theory and Practice*. McGraw-Hill, United States of America, 7th edition, 1998. ISBN 0-07-015914-9.

[8] P. F Fisher and N. J. Tate. Causes and consequences of error in digital elevation models. *Progress in Physical Geography*, 30:467–489, 2006.

[9] G. L. Heritage, D. J. Milan, A. R. G. Large, and I. C. Fuller. Influence of survey strategy and interpolation model on DEM quality. *Geomorphology*, 112(3âĂŞ4): 334–344, 2009. ISSN 0169-555X. doi: http://dx.doi.org/10.1016/j.geomorph.2009.06.024. URL http://www.sciencedirect.com/science/article/pii/S0169555X09002591.

[10] S. G. Bangen. *Comparison of Topographic Surveying Techniques in Streams*. Thesis, Utah State University, 2013.

[11] K. Unal, S. Ercan, and Y. Erkan. Performance and accuracy comparisons of GPS and total station in land surveying. In *2006 ASABE Annual International Meeting*, 2006. doi: 10.13031/2013.20581. URL http://elibrary.asabe.org/abstract.asp?aid=20581&t=5.

[12] Pratik Agarwal, Wolfram Burgard, and Cyrill Stachniss. Survey of geodetic mapping methods: Geodetic approaches to mapping and the relationship to graph-based SLAM. *IEEE Robotics & Automation Magazine*, 21(3):63–80, 2014.

[13] Land Information New Zealand. Datums, projections and heights. https://www.linz.govt.nz/data/geodetic-system/datums-projections-and-heights, . Accessed: 2020-04-13.

[14] Land Information New Zealand. World geodetic system 1984 (wgs84). https://www.linz.govt.nz/data/geodetic-system/datums-projections-and-heights/geodetic-datums/world-geodetic-system-1984-wgs84, . Accessed: 2020-04-13.

[15] National Oceanic and Atmospheric Administration (NOAA). What is the geoid? https://oceanservice.noaa.gov/facts/geoid.html. Accessed: 2020-04-13.

[16] Z. Li, Q. Zhu, and C. Gold. *Digital Terrain Modelling: Principles and Methodology*. CRC Press, 2004. ISBN 0415324629.

[17] Z. Li. *Sampling strategy and accuracy assessment for digital terrain modelling*. Thesis, University of Glasgow, 1990.

[18] W. Fraczek. Mean sea level, GPS, and the geoid. http://www.esri.com/news/arcuser/0703/geoid1of3.html. Accessed: 2018-06-01.

[19] S. Lane, K. Richards, and J. Chandler. *Landform Monitoring, Modelling and Analysis*. John Wiley & Sons Ltd, 1998. ISBN 0-471-96977-X.

[20] V. J. Sickle. *GPS for Land Surveyors*. Ann Arbor Press, 1996. ISBN 1-57504-041-7.

[21] United States Government. Global Positioning System Standard Positioning Service Performance Standard. Government document, United States Government, 2008. URL http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf.

[22] W. Schofield and M. Breach. *Engineering Surveying*. CRC Press, Oxford, 6th edition, 2007. ISBN 9781136406065. URL http://canterbury.eblib.com.au/patron/FullRecord.aspx?p=294118.

[23] A. Leick, L. Rapoport, and D. Tatarnikov. *GPS Satellite Surveying*. Wiley, 4th edition, 2015. ISBN 9781118675571.

[24] Trimble Inc. Trimble Access General Survey. Report, Trimble Inc, 2015. URL http://apps.trimbleaccess.com/help/en/TrimbleAccess=2015.22.

[25] R. Graham and A. Koh. *Digital Aerial Survey: Theory and Practice*. Whittles Publishing, 2002. ISBN 978-1-84995-085-5.

[26] Ian L. Turner, Mitchell D. Harley, and Christopher D. Drummond. UAVs for coastal surveying. *Coastal Engineering*, 114:19 – 24, 2016. ISSN 0378-3839. doi: http://dx.doi.org/10.1016/j.coastaleng.2016.03.011. URL http://www.sciencedirect.com/science/article/pii/S0378383916300370.

[27] P. R. Wolf and B. A. Dewitt. *Elements of photogrammetry: with applications in GIS*, volume 3. McGraw-Hill New York, 2000.

[28] M. Gerke and F. Remondino. Oblique airborne photogrammetry: usersâĂŹ and vendorsâĂŹ views. *GIM Int.*, pages 18–19, 2014.

[29] G. J. Grenzdörffer, M. Guretzki, and I. Friedlander. Photogrammetric image acquisition and image analysis of oblique imagery. *The Photogrammetric Record*, 23 (124):372–386, 2008.

[30] S. N. Lane, K. S. Richards, and J. H. Chandler. Developments in monitoring and modelling small-scale river bed topography. *Earth Surface Processes and Landforms*, 19(4):349–368, 1994. ISSN 1096-9837. doi: 10.1002/esp.3290190406. URL http://dx.doi.org/10.1002/esp.3290190406.

[31] X. Liu, Z. Zhang, J. Peterson, and S. Chandra. The Effect of LiDAR Data Density on DEM Accuracy. In *Proceedings of International Congress on Modelling and Simulation (MODSIM07)*, pages 1363–1369, December 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.4833&rep=rep1&type=pdf.

[32] Y. Jia, T. Lan, T. Peng, H. Wu, C. Li, and G. Ni. Effects of point density on dem accuracy of airborne LiDAR. In *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, pages 493–496, 2013. ISBN 2153-6996. doi: 10.1109/IGARSS.2013.6721200.

[33] University of York. Topographic survey, 2016. URL http://www.york.ac.uk/media/archaeology/documents/undergraduatedegrees/guides/ts.pdf.

[34] M. J. Smith, M. F. Goodchild, and P. A. Longley. *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. Matador, 2007. ISBN 1-905886-60-8.

[35] P. V. Arun. A comparative analysis of different dem interpolation methods. *The Egyptian Journal of Remote Sensing and Space Science*, 16(2):133–139, 2013. ISSN

1110-9823. doi: http://dx.doi.org/10.1016/j.ejrs.2013.09.001. URL http://www.sciencedirect.com/science/article/pii/S1110982313000276.

[36] V. Chaplot, F. Darboux, H. Bourennane, S. Leguédois, N. Silvera, and K. Phachomphon. Accuracy of interpolation techniques for the derivation of digital elevation models in relation to landform types and data density. *Geomorphology*, 77(1âĂŞ2):126–141, 2006. ISSN 0169-555X. doi: http://dx.doi.org/10.1016/j.geomorph.2005.12.010. URL http://www.sciencedirect.com/science/article/pii/S0169555X06000079.

[37] J. D. Wood and P. F. Fisher. Assessing interpolation accuracy in elevation models. *IEEE Computer Graphics and Applications*, 13(2):48–56, March 1993. ISSN 0272-1716. doi: 10.1109/38.204967.

[38] D. C. Dauwalter, W. L. Fisher, and K. C. Belt. Mapping stream habitats with a global positioning system: Accuracy, precision, and comparison with traditional methods. *Environmental Management*, 37(2):271–280, 2006. ISSN 1432-1009. doi: 10.1007/s00267-004-0270-z. URL http://dx.doi.org/10.1007/s00267-004-0270-z.

[39] P. C. Kyriakidis, A. M. Shortridge, and M. F. Goodchild. Geostatistics for conflation and accuracy assessment of digital elevation models. *International Journal of Geographical Information Science*, 13(7):677–707, 1999. ISSN 1365-8816. doi: 10.1080/136588199241067. URL http://dx.doi.org/10.1080/136588199241067.

[40] G. Xu and Y. Xu. *GPS: Theory, Algorithms and Applications*. Springer, 3rd edition, 2016. ISBN 978-3-662-50367-6.

[41] A. L. Allan. *Principles of Geospatial Surveying*. Whittles Publishing, November 2007. ISBN 978-1-849-95145-6.

[42] J. Brasington, B. T. Rumsby, and R. A. McVey. Monitoring and modelling morphological change in a braided gravel-bed river using high resolution GPS-based survey. *Earth Surface Processes and Landforms*, 25(9):973–990, 2000. ISSN 1096-9837. doi: 10.1002/1096-9837(200008)25:9<973::AID-ESP111>3.0.CO;2-Y. URL http://dx.doi.org/10.1002/1096-9837(200008)25:9<973::AID-ESP111>3.0.CO;2-Y.

[43] F. Dai, Y. Feng, and R. Hough. Photogrammetric error sources and impacts on modeling and surveying in construction engineering applications. *Visualization in Engineering*, 2(1):2, 2014. ISSN 2213-7459. doi: 10.1186/2213-7459-2-2. URL

http://dx.doi.org/10.1186/2213-7459-2-2.

[44] P. Teunissen and O. Montenbruck. *Springer handbook of global navigation satellite systems*. Springer, 2017.

[45] Trimble Inc. Trimble GPS tutorial - error correction. http://www.trimble.com/gps_tutorial/howgps-error.aspx, Jan 2017. Accessed: 2018-04-10.

[46] NovAtel. Chapter 4: GNSS error sources. http://www.novatel.com/an-introduction-to-gnss/chapter-4-gnss-error-sources/error-sources/. Accessed: 2019-11-04.

[47] R. B. Langley. Dilution of precision. *GPS World*, pages 52–59, May 1999. URL http://gauss.gge.unb.ca/papers.pdf/gpsworld.may99.pdf.

[48] OneSky. How "good" will GPS be when you fly? https://onesky.blog/2017/08/21/how-good-will-gps-be-when-you-fly%EF%BB%BF/. Accessed: 2020-04-11.

[49] D. P. Paine and J. D Kiser. *Aerial Photography and Image Interpretation*. Wiley, 3rd edition, 2012. ISBN 978-1-118-11262-5.

[50] J. Meguro, T. Murata, J. Takiguchi, Y. Amano, and T. Hashizume. GPS multipath mitigation for urban area using omnidirectional infrared camera. *IEEE Transactions on Intelligent Transportation Systems*, 10(1):22–30, March 2009. ISSN 1558-0016. doi: 10.1109/TITS.2008.2011688.

[51] S. A. Weaver, Z. Ucar, P. Bettinger, and K. Merry. How a gnss receiver is held may affect static horizontal position accuracy. *Plos One*, 10(4), April 2015.

[52] M. Brach and Michał Z. The effect of mounting height on gnss receiver positioning accuracy in forest conditions. *Croatian Journal of Forest Engineering: Journal for Theory and Application of Forestry Engineering*, 35(2):245–253, 2014.

[53] J. G. Fryer, J. H. Chandler, and M. A. R. Cooper. On the accuracy of heighting from aerial photographs and maps: Implications to process modellers. *Earth Surface Processes and Landforms*, 19(6):577–583, 1994. ISSN 1096-9837. doi: 10.1002/esp.3290190609. URL http://dx.doi.org/10.1002/esp.3290190609.

[54] U.S. Army Corps of Engineers. Handbook for transformation of datums, projections, grids and common coordiante systems. Technical report, U.S. Army Corps of Engineers, January 1996.

[55] A. Hornung, K. M Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. URL http://octomap.github.com.

[56] I. Parra-Tsunekawa, J. Ruiz-del Solar, and P. Vallejos. A kalman-filtering-based approach for improving terrain mapping in off-road autonomous vehicles. *Journal of Intelligent & Robotic Systems*, 78(3):577–591, 2015. ISSN 1573-0409. doi: 10.1007/s10846-014-0087-9. URL http://dx.doi.org/10.1007/s10846-014-0087-9.

[57] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[58] P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 26:217–230, 2007.

[59] B. Douillard, J. Underwood, N. Melkumyan, S. Singh, S. Vasudevan, C. Brunner, and A. Quadros. Hybrid elevation maps: 3D surface models for segmentation. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1532–1538, 2010.

[60] D. Meagher. Geometric modelling using octree encoding. *Computer Graphics and Modelling*, 19(2):129–147, 1981.

[61] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, July 1992. ISSN 0730-0301. doi: 10.1145/130881.130882. URL http://doi.acm.org/10.1145/130881.130882.

[62] P. Payeur, P. Hebert, D. Laurendeau, and C. M. Gosselin. Probabilistic octree modeling of a 3D dynamic environment. In *Proceedings of International Conference on Robotics and Automation*, volume 2, pages 1289–1296 vol.2, Apr 1997. doi: 10.1109/ROBOT.1997.614315.

[63] A. Hornung, M. Phillips, E. Gil Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. *2012 IEEE International Conference on Robotics and Automation*, pages 423–429, 2012.

[64] J. Fournier, B. Ricard, and D. Laurendeau. Mapping and exploration of complex environments using persistent 3D model. In *Computer and Robot Vision, 2007. CRV '07. Fourth Canadian Conference on*, pages 403–410, May 2007. doi: 10.1109/CRV.2007.45.

[65] D. M. Cole and P. M. Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proceedings 2006 IEEE International Conference on Robotics*

and Automation, 2006. ICRA 2006., pages 1556–1563, May 2006. doi: 10.1109/ROBOT.2006.1641929.

[66] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[67] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM-3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007. ISSN 1556-4967. doi: 10.1002/rob.20209. URL http://dx.doi.org/10.1002/rob.20209.

[68] D. Ferguson, A. Morris, D. Hahnel, C. Baker, Z. Omohundro, C. Reverte, S. Thayer, W. Whittaker, W. Whittaker, W. Burgard, and S. Thrun. An autonomous robotic system for mapping abandoned mines. *IEEE Robotics & Automation Magazine*, 11(4):79–91, 2004. ISSN 1070-9932. doi: 10.1109/MRA.2004.1371614.

[69] J. Zhang and S. Singh. LOAM: LiDAR odometry and mapping in real-time. In *Robotics: Science and Systems Conference*, July 2014.

[70] J. Razlaw, D. Droeschel, D. Holz, and S. Behnke. Evaluation of registration methods for sparse 3D laser scans. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–7, Sept 2015. doi: 10.1109/ECMR.2015.7324196.

[71] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of surface normal estimation methods for range sensing applications. In *2009 IEEE International Conference on Robotics and Automation*, pages 3206–3211, May 2009. doi: 10.1109/ROBOT.2009.5152493.

[72] K. Jordan and P. Mordohai. A quantitative evaluation of surface normal estimation in point clouds. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4220–4226, Sept 2014. doi: 10.1109/IROS.2014.6943157.

[73] S. Jin, R. R. Lewis, and D. West. A comparison of algorithms for vertex normal computation. *The Visual Computer*, 21(1):71–82, Feb 2005. ISSN 1432-2315. doi: 10.1007/s00371-004-0271-1. URL https://doi.org/10.1007/s00371-004-0271-1.

[74] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 71–78, New York, NY, USA, 1992. ACM. ISBN 0-89791-479-1. doi: 10.1145/133994.134011.

URL http://doi.acm.org/10.1145/133994.134011.

[75] S. Gumhold, X. Wang, and R. Macleod. Feature extraction from point clouds. In *In Proceedings of the 10 th International Meshing Roundtable*, pages 293–305, 2001.

[76] F. Tombari, S. Salti, and L. Di Stefano. *Unique Signatures of Histograms for Local Surface Description*, pages 356–369. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15558-1. doi: 10.1007/978-3-642-15558-1_26. URL https://doi.org/10.1007/978-3-642-15558-1_26.

[77] N. J Mitra and A. Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 322–328. ACM, 2003.

[78] R. Hoffman and A. K. Jain. Segmentation and classification of range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):608–620, Sept 1987. ISSN 0162-8828. doi: 10.1109/TPAMI.1987.4767955.

[79] C. Wang, H. Tanahashi, H. Hirayu, Y. Niwa, and K. Yamamoto. Comparison of local plane fitting methods for range data. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, Dec 2001. doi: 10.1109/CVPR.2001.990538.

[80] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. ISSN 0162-8828. doi: 10.1109/34.121791.

[81] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proccedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729, 1991.

[82] A. V. Segal, D. Hähnel, and S. Thrun. Generalized ICP. In *Robotics: Science and Systems V*, June 2009. doi: 10.15607/RSS.2009.V.021.

[83] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. doi: 10.1109/IM.2001.924423.

[84] M. Greenspan and M. Yurick. Approximate k-d tree search for efficient ICP. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 442–448, Oct 2003. doi: 10.1109/IM.2003.1240280.

[85] D. A. Simon. *Fast and Accurate Shape-Based Registration*. PhD thesis, Carnegie Mellon University, December 1996.

[86] D. Holz and S. Behnke. Registration of non-uniform density 3D point clouds using approximate surface reconstruction. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–7. VDE, 2014.

[87] D. Holz and S. Behnke. Mapping with micro aerial vehicles by registration of sparse 3D laser scans. In *13th Internation Conference on Intelligent Autonomous Systems*, 2014.

[88] J. Zhang and S. Singh. Low-drift and real-time LiDAR odometry and mapping. *Autonomous Robots*, pages 1–16, 2016.

[89] Diego Viejo and Miguel Cazorla. 3D plane-based egomotion for SLAM on semi-structured environment. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2761–2766. IEEE, 2007.

[90] J. Xiao, B. Adler, and H. Zhang. 3D point cloud registration based on planar surfaces. In *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 40–45, Sep. 2012. doi: 10.1109/MFI.2012.6343035.

[91] W. S. Grant, R. C. Voorhies, and L. Itti. Finding planes in LiDAR point clouds for real-time registration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4347–4354. IEEE, 2013.

[92] K. Pathak, A. Birk, N. Vaskevicius, M. Pfingsthorn, S. Schwertfeger, and J. Poppinga. Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation. *Journal of Field Robotics*, 27(1): 52–84, 2010.

[93] K. Georgiev, R. T. Creed, and R. Lakaemper. Fast plane extraction in 3D range data based on line segments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3808–3815. IEEE, 2011.

[94] O. Wulf, A. Nüchter, J. Hertzberg, and B. Wagner. Benchmarking urban six-degree-of-freedom simultaneous localization and mapping. *Journal of Field Robotics*, 25(3):148–163, 2008. ISSN 1556-4967. doi: 10.1002/rob.20234. URL http://dx.doi.org/10.1002/rob.20234.

[95] D. Borrmann, J. Elseberg, K. Lingermann, A. Nüchter, and J. Hertzberg. The efficient extension of globally consistent scan matching to 6 DOF. *Knowledge Based Systems*, 1:20, 2008.

[96] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005. ISBN 978-0-262-20162-9.

[97] T. D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[98] L. Perea, J. How, L. Breger, and P. Elosegui. Nonlinearity in sensor fusion: divergence issues in EKF, modified truncated GSF, and UKF. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6514, 2007.

[99] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000. doi: 10.1109/ASSPCC.2000.882463.

[100] J. K. Uhlmann. *Simaltaneous Map Building and Localization for Real Time Applications*. PhD thesis, 1994.

[101] S. J. Julier and J. K. Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[102] M. Bozorg, M. S. Bahraini, and A. B. Rad. New adaptive UKF algorithm to improve the accuracy of SLAM. *International Journal of Robotics, Theory and Applications*, 5(1):35–46, 2019.

[103] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*. Springer, 2016.

[104] P. Corke. *Robotics, vision and control: fundamental algorithms in MATLAB second, completely revised*, volume 118. Springer, 2017.

[105] P. Cheeseman, R. Smith, and M. Self. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotic Research*, pages 467–474, 1987.

[106] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer, 1990.

[107] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National conference on Artificial Intelligence*, pages 593–598, 2002.

[108] J. J. Leonard and H. J. S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Robotics Research*, pages 169–176. Springer, 2000.

[109] S. Williams, G. Dissanayake, and H. Durrant-Whyte. Efficient simultaneous localisation and mapping using local submaps. In *Workshop Notes of the ICRA Workshop on Concurrent Mapping and Localization for Autonomous Mobile Robots (W4), Washington, DC*, 2002.

[110] P. Piniés and J. D. Tardós. Scalable SLAM building conditionally independent local maps. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3466–3471. IEEE, 2007.

[111] L. M. Paz, J. D. Tardós, and J. Neira. Divide and conquer: EKF SLAM in $o(n)$. *IEEE Transactions on Robotics*, 24(5):1107–1120, 2008.

[112] S. Huang, Z. Wang, and G. Dissanayake. Sparse local submap joining filter for building large-scale maps. *IEEE Transactions on Robotics*, 24(5):1121–1130, 2008.

[113] Z. Gong, J. Li, Z. Luo, C. Wen, C. Wang, and J. Zelek. Mapping and semantic modeling of underground parking lots using a backpack lidar system. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 12 2019. doi: 10.1109/TITS.2019.2955734.

[114] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1985–1991. IEEE, 2003.

[115] J. A. Castellanos, J. Neira, and J. D. Tardós. Limits to the consistency of EKF-based SLAM. *IFAC Proceedings Volumes*, 37(8):716–721, 2004.

[116] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 4(3):380–407, 2004.

[117] C. Stachniss, G. Grisetti, W. Burgard, and N. Roy. Analyzing gaussian proposal distributions for mapping with rao-blackwellized particle filters. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3485–3490. IEEE, 2007.

[118] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, pages 1151–1156, 2003.

[119] Z. Kurt-Yavuz and S. Yavuz. A comparison of EKF, UKF, fastSLAM2. 0, and UKF-based fastSLAM algorithms. In *2012 IEEE 16th International Conference on Intelligent Engineering Systems (INES)*, pages 37–43. IEEE, 2012.

[120] R. Sim, P. Elinas, M. Griffin, A. Shyr, and J. J. Little. Design and analysis of a framework for real-time vision-based SLAM using rao-blackwellised particle filters. In *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, pages

21–21. IEEE, 2006.

[121] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. TardÃűs. A comparison of SLAM algorithms based on a graph of relations. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2089–2095, Oct 2009. doi: 10.1109/IROS.2009.5354691.

[122] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *The International Journal of Robotics Research*, 23(12):1113–1139, 2004. doi: 10.1177/0278364904049393. URL http://dx.doi.org/10.1177/0278364904049393.

[123] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An atlas framework for scalable mapping. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1899–1906 vol.2, Sept 2003. doi: 10.1109/ROBOT.2003.1241872.

[124] H. Kretzschmar and C. Stachniss. Information-theoretic compression of pose graphs for laser-based SLAM. *The International Journal of Robotics Research*, 31(11):1219–1230, 2012. doi: 10.1177/0278364912455072. URL http://dx.doi.org/10.1177/0278364912455072.

[125] E. Nelson. Berkley localization and mapping, 2016. URL https://github.com/erik-nelson/blam.

[126] C. Roussillon, A. Gonzalez, J. Solà, J. Codol, N. Mansard, S. Lacroix, and M. Devy. Rt-SLAM: a generic and real-time visual SLAM implementation. In *International Conference on Computer Vision Systems*, pages 31–40. Springer, 2011.

[127] C. Roussillon and S. Lacroix. High rate-localization for high-speed all-terrain robots. In *CCCA12*, pages 1–8. IEEE, 2012.

[128] M. Labbé and F. Michaud. RTAB-map as an open-source LiDAR and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[129] R Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

[130] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):

1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[131] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[132] C. Stachniss G. Grisetti and W. Burgard. Gmapping, 2019. URL https://openslam-org.github.io/gmapping.html.

[133] Clearpath Robotics. Jackal small unmanned ground vehicle, Dec 2016. URL https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/.

[134] M. Purvis and T. Baltovski. Jackal, 2019. URL https://github.com/jackal.

[135] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[136] Tom Moore. State estimation nodes. http://docs.ros.org/melodic/api/robot_localization/html/state_estimation_nodes.html?highlight=tuning. Accessed: 2020-05-05.

[137] F. A. Donoso, K. J. Austin, and P. R. McAree. How do ICP variants perform when used for scan matching terrain point clouds? *Robotics and Autonomous Systems*, 87(Supplement C):147 – 161, 2017. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2016.10.011. URL http://www.sciencedirect.com/science/article/pii/S0921889016301282.

[138] R. B. Rusu and S. Cousins. 3D is here: Point cloud library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011. doi: 10.1109/ICRA.2011.5980567.

[139] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, and S. Boldo. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[140] V. Rajaraman. IEEE standard for floating point numbers. *Resonance*, 21(1):11–30, Jan 2016. ISSN 0973-712X. doi: 10.1007/s12045-016-0292-x. URL https://doi.org/10.1007/s12045-016-0292-x.

[141] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, Apr 2013. ISSN 1573-7527. doi: 10.1007/s10514-013-9327-2. URL https://doi.org/10.1007/s10514-013-9327-2.

[142] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[143] The stanford 3D scanning repository, 2019. URL http://graphics.stanford.edu/data/3Dscanrep/.

[144] K. Low. Linear least-squares optimization for point-to-plane ICP surface registration. Technical report, University of North Carolina, 2004.

[145] J. Sprickerhof, A. Nüchter, K. Lingemann, and J. Hertzberg. An explicit loop closing technique for 6d SLAM. In *ECMR*, pages 229–234, 2009.

[146] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[147] S. G. Johnson. The NLopt nonlinear-optimization package, 2020. URL http://github.com/stevengj/nlopt.

[148] S. G. Johnson. NLopt algorithms, 2020. URL https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/.

[149] R. P. Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[150] M. J. D. Powell. Advances in optimization and numerical analysis. In *Proceeding of the 6th Workshop on Optimization and Numerical Analysis*, pages 5–67, 1994.

[151] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[152] T. H. Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, 1991.

[153] M. Powell. The newuoa software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer, 2006.

[154] Trimble Inc. Trimble SX10 scanning total station datasheet. Report, Trimble Inc, 2019. URL https://geospatial.trimble.com/sites/default/files/2019-10/Datasheet%20-%20SX10%20Scanning%20Total%20Station%20-%20English%20A4%20-%20Screen.pdf.

[155] Velodyne LiDAR Inc. Velodyne LiDAR HDL-32E. Datasheet, 2018. URL https://velodynelidar.com/products/hdl-32e/.

[156] W. Meeussen. Rep 105 - coordinate frames for mobile platforms. http://www.ros.org/reps/rep-0105.html, Dec . Accessed: 2017-02-16.

[157] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference*

on Intelligent Autonomous Systems (IAS-13). Springer, July 2014.

A | Appendix B: Hardware and Software Specifications

A.1 Hardware

All experiments were conducted on a Dell Precision M3800 laptop with an Intel Core i7-4712HQ 2.30 GHz processor running an Ubuntu 14.04 operating system.

The basis of the robot chassis was a Clearpath Robotics "Jackal". It was upgraded with an InvenSense MPU-9250A Iner-tial Measurement Unit (IMU), Trimble R7 GNSS receiver, Trimble Zephyr 2 antenna and Velodyne HDL-32E. The data sheets for each device can be found on the manufacturers website .

A.2 Software

Table A.1 details the software specifications of each third-party library and software package used in this thesis. In each case, the most recent working version of each library or package was used. Because the Clearpath Jackal shipped with ROS Indigo and Ubuntu 14.04, this necessitated using the same versions on the laptop, which in some cases restricted the latest version of some packages that could be used. Where necessary, packages were installed from source rather than pre-compiled binaries.

Software package	**Version**
Operating System	Ubuntu 14.04
Robot Operating System (ROS)	Indigo Igloo 1.11.21
robot_localization	2.3.4
Point Cloud Library (PCL)	1.8.1
Google Cartographer	
Trimble Business Center	5.10
NLopt	2.6.1
Eigen	3.2.0
C++	C++11
Cmake	
Qmake	
GCC	4.8.4
G++	4.8.4
Qt	

TABLE A.1: Software specifications

B | Appendix B: Robot Prototype System Configuration

B.1 Localization Software Setup

Robot localization is the process of determining where a robot is within its environment. More specifically, calculating the position and orientation of the robot relative to some local or global coordinate system. There wide variety of tools and methods available for doing this, and the literature on robot localization is vast.

But within the ROS ecosystem, localization of a robot is defined by the ROS Enhancement Proposals (REPs), which are operating standards that all ROS packages are required to adhere to in order to maintain compatibility. The transforms between these frames are typically generated by localization packages, such as `robot_localization` or `gmapping`, a 2D laser-based SLAM system.

This section will briefly discuss the key reference frames, why they are important, and how they are used by the KFs in the `robot_localization` package to accurately localize the UGV in an outdoor environment.

B.1.1 Reference Frames

A key part of outdoor localization is localizing the robot in a reference frame that is accurate over long distance. Several ROS REPs are important for localization, and specify how reference frames should be linked and what they should represent. In particular, REP 105 specifies that every robotic localization package should use the following three frames:

1. **`base_link`** - This reference frame is rigidly fixed to the body of the robot and represents the location and orientation of the robot. Reference frames for any

other part of the robot (wheels, sensors, hardpoints) are generally defined relative to `base_link`.

2. **`odom`** - This is a short-term, world-fixed reference frame. The robot's position in this frame (i.e. the position of `base_link` relative to `odom`) is generally calculated using data from wheel encoders, IMUs and visual odometry. This way the position of the robot should be smooth and continuous, without any discrete jumps [156]. This makes the `odom` reference frame ideal for local navigation. Because these sensors all provide relative rather than absolute data, the position error accumulates which makes the `odom` frame inaccurate over long distances.

3. **`map`** - This is a long-term, world-fixed reference frame. The robot's position in this frame is calculated using GNSS data as well as any other sensor data available. The inclusion of absolute position data from the GNSS makes the robot's position in this reference frame accurate over very long distances. Data in this frame may also include discrete jumps, depending on the setup and sensor quality.

Understanding these reference frames and the differences between them is important as they will be referred to repeatedly in this chapter. The `robot_ localization` package adheres to REP 105, and is designed so that a typical setup uses KFs to create and link all three frames. Each KF is created and run by a separate instance of the `ekf_localization` or `ukf_localization` node. To fuse GNSS data an instance of the `navsat_transform` node is also required. Together, these three ROS nodes form the basis of the localization module for this autonomous UGV. The remainder of this section will explain how each node is used to link the three frames.

The `base_link` reference frame, and any child frames, are defined in the robot's Unified Robot Description Format (URDF) file. For the Jackal chassis, this file is pre-defined and provided by Clearpath Robotics. The links between each frame in the URDF are broadcast by the `robot_state_publisher` node, which runs on boot up. Each sensor and major component of the robot has it's own reference frame, such as each of the wheels (`wheel_link`), the IMU (`imu_link`) and mounted R7 GNSS receiver (`auxgps_link`).

The `odom` reference frame is created by the first "local" state estimation filter. This filter fuses data from the IMU and wheel encoders and then outputs the position of the

`base_link` relative to `odom`. The origin of the `odom` frame is set at the robot's current position, at the time the local `ekf_localization` node is launched.

The `map` reference frame is created by a second "global" state estimation filter. This filter is set up to fuse GNSS, IMU and wheel encoder data in the `map` reference frame. The addition of GNSS data makes the output of this filter accurate over long distances. Because ROS conventions state that a reference frame can only have one parent [156], the second filter links `map` to `base_link` via the `odom` frame. And like the `odom` frame, the `map` frame is set at the robots position at the time the global `ekf_localization` node is launched.

If fusing GNSS data, an instance of the `navsat_transform` node must be set up to convert WGS84 latitude and longitude coordinates from the GNSS receiver into an odometry pose relative to the `map` reference frame. The node achieves this by creating an additional reference frame called `utm` which is centered at the 0,0 point of the UTM grid. `navsat_transform` then uses a series of frame transforms between this reference frame and the `base_link`/`map` reference frames [157]. The `utm` frame is defined as a child of the `map` frame.

This produces a frame hierarchy like the simplified version shown in figure B.1, while figure B.2 shows the physical relationship between the key frames. Note that figure B.2 does not show the location of the `utm` frame, as it is located thousands of meters from the UGV. At the time the `ekf_localization` nodes are launched, the `base_link`, `odom` and `map` frames overlap and have identical origins. Figure B.2 shows the robot after it has moved from this origin.

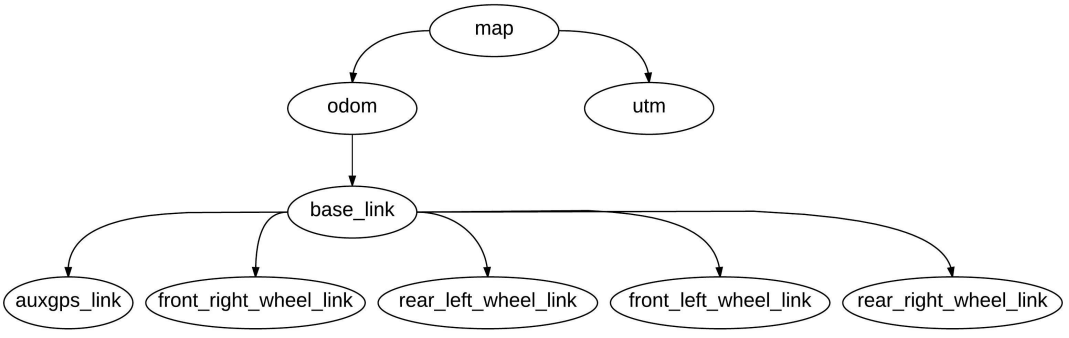


FIGURE B.1: Simplified version of the UGV reference frame tree

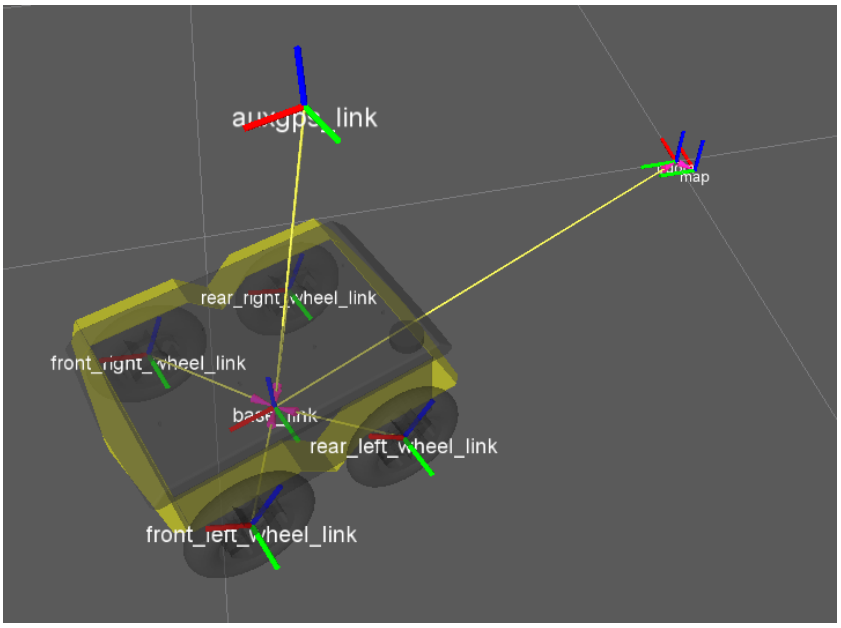


FIGURE B.2: Physical location of the key ROS reference frames

Put together, the frames `base_link`, `odom`, `map` and `utm` localize the robot. The `odom` and `map` frames provide a reference for the UGV's postion relative to its origin, while the `utm` frame provides a reference for the absolute global position of the UGV.

B.1.2 State Estimation Configuration

The KFs in the `robot_localization` package can be used to localize the robot relative to a origin point (`odom` or `map`). However the type and the setup of the filters is important to the accuracy of the filter and how well it performs in different environments.

The `robot_localization` package can implement either an Extended or Unscented Kalman Filter. The UKF is generally more stable than the EKF, but it is also more computationally expensive. Because of the additional sigma parameters, the UKF is also harder to tune and optimize. In terms of accuracy, the two produce similar results in linear systems, and it is only in highly non-linear systems that the UKF becomes significantly more accurate. For these reasons, EKFs are more widely used and are often considered the *de facto* robot localization algorithm [96]. Both an EKF and a UKF were used on the first robotic prototype, and were used to localize the robot for the work described in Chapter 6. That research showed that the UKF was consistently more accurate at localizing the robot, and as result the second prototype was configured to use UKFs exclusively.

The `robot_localization` node implements an EKF or UKF as described in [157]. Template configuration files are provided and these were used with few changes. However some changes were made and some configuration parameters should be described to assist the reader in understanding what sensor data is being fused and how.

Each filter estimates the 15-dimensional state of the robot: $(X, Y, Z,\ roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z})$. Note that the KF does not build a map or directly localize the robot within one. It simply provides an estimate of the robot's state relative to a particular reference frame. The configuration files expose a parameter for determining which variables from a sensor will be fused in the final state estimate. The parameter is a matrix of the form:

$$\begin{bmatrix} X & Y & Z \\ roll & pitch & yaw \\ \dot{X} & \dot{Y} & \dot{Z} \\ \dot{roll} & \dot{pitch} & \dot{yaw} \\ \ddot{X} & \ddot{Y} & \ddot{Z} \end{bmatrix}$$

Where each value is fused if the corresponding matrix entry is set to "`true`". As an example, the default configuration for a wheel encoder odometry source is:

$$\begin{bmatrix} False & False & False \\ False & False & False \\ True & True & True \\ False & False & True \\ False & False & False \end{bmatrix}$$

So only the $\dot{X}, \dot{Y}, \dot{Z}$ and $\dot{yaw}$ components of the wheel encoder data will be fused into the state estimate. This parameter ensures that only relevant information from each sensor is fused into the state estimate. In this research, the recommended data was fused for each sensor as listed here:

- **Wheel encoder data** - $\dot{X}, \dot{Y}, \dot{Z},\ \dot{yaw}$.
- **IMU data** - $roll, pitch, yaw,\ \dot{roll}, \dot{pitch}, \dot{yaw}$.

- **GNSS data** - X, Y, Z

Several other parameters exist for optimizing the performance of the implemented EKF, such as adjusting thresholds on sensor data, the process noise and the initial state estimate noise. These parameters allow the user to control how quickly the filter converges to a solution, and which data is relied on more heavily in the state estimate. In general, these parameters provide superior performance if well tuned, but tuning them manually can be difficult. Because of the difficulty in tuning an EKF or UKF, this was deemed to be outside the scope of this project.

B.2 Navigation Software Setup

Navigation is the next key component of autonomous robotics. In ROS this is achieved using the Navigation Stack, a collection of localization, SLAM, map and path-planning functions. As previously explained in section 5.2.2, the ROS navigation stack was chosen to act as the basis for autonomous navigation for the UGV.

The following section explains how autonomous navigation with the UGV was achieved by building up the core functionality in three stages. First, by achieving autonomous navigation between a current and desired location (Section B.2.1: Single-goal Path-following). Second by expanding this to navigate between multiple sequential waypoints (section B.2.2: Autonomous Waypoint Navigation). Third, by adding obstacle avoidance with Gmapping 2D SLAM.

B.2.1 Single-goal Path-following

This section briefly describes how the `move_base` node autonomously navigates between points and how it was configured for outdoor use. With reference to figure B.3, the core of the navigation stack is the `move_base` node which manages two key processes:

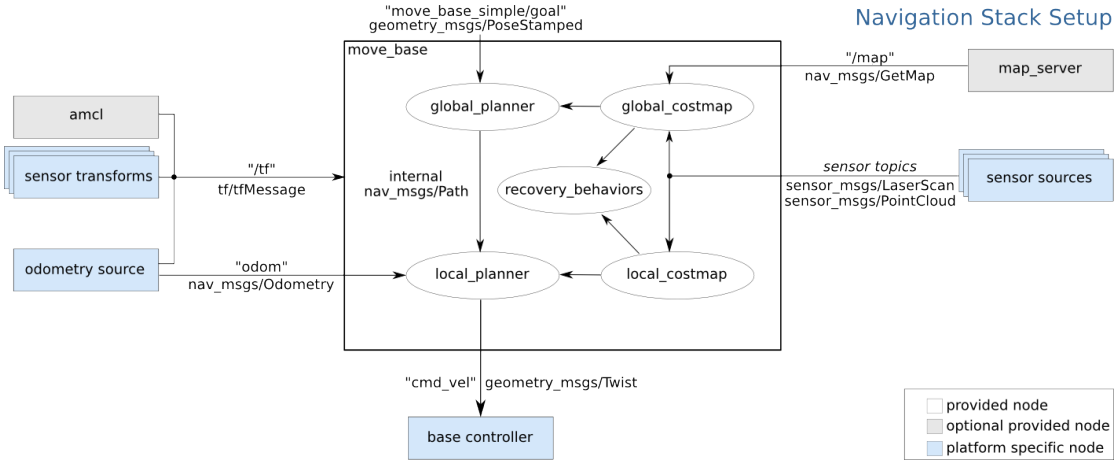


FIGURE B.3: Overview of the ROS navigation stack

Source: wiki.ros.org

1. **`global_planner`** - The global planner is responsible for navigating the robot from it's starting location to a desired location (called a move goal). It uses position data from a fixed global reference frame such as `map` to plot a path to the destination through any real-world obstacles, which are represented in a global cost map. The default algorithm is Dijkstra's A.

2. **`local_planner`** - The local planner is responsible for moving the robot along the global plan whilst avoiding nearby obstacles. It uses position data from a local reference frame such as `odom` and any nearby obstacles are represented in a local cost map. Note that it is the local, not global, planner that issues move commands to the robot's motor controllers. The default algorithm is Dynamic Window Approach.

When these planners are used together, the `move_base` node allows the user to specify a move goal in any reference frame which the robot will then autonomously navigate to in the real world.

The `move_base` node also allows the user to configure a variety of parameters, from the size and resolution of each cost map, to the maximum and minimum velocities of robot. Clearpath Robotics provides a number of recommended configuration files for the navigation stack, which have been use for this project with few deviations. The files create an instance of the navigation stack as illustrated in Figure B.3, with a blank costmap and odometry provided by the EKFs or UKFs previously described in Section

B.1. However, there are some key changes that have been made to allow the `move_base` node to navigate accurately over long distances.

The `move_base` node is designed for short, indoor use. So by default any move goals issued to it will be converted into the local `odom` frame, and the node will attempt to reach the goal *in that reference frame*. But because of the inherent position drift in this frame, the goal in the `map` frame and the same goal in the `odom` frame will be in different positions. The key changes made to the `move_base` node are as follows:

Costmap Size.

The default `global_costmap` size is 30x30 m. Because the global path planner operates on this map it cannot plot a path to a point outside it. 30x30 m is enough for indoor navigation but not for long-distance navigation outdoor. Therefore the size was increased to 80x80 m. The `local_costmap` was configured by default as a "rolling window" i.e. the map is not fixed and moves so that the UGV is always at it's center.

Costmap Global frame

When a move goal is issued to the `move_base` node, it is converted into the reference frame of the `local_costmap` and the `move_base` will attempt to navigate the robot to the goal *in that reference frame*. This parameter defaults to the `odom` reference frame. The problem is, as previously discussed in section B.1.1, over long distances the odometry in the `odom` frame drifts. So after travelling long distances the position of a move goal in the `odom` frame will have drifted from its real-world location in the `map` frame. As a result, this parameter must be changed from `odom` to `map` in order to make autonomous navigation over long distances accurate.

Path-planner Frequency

The rate at which the `global_planner` re-plots the global path is determined by the `planner_ frequency` parameter. The default value in Clearpath configuration files is 20 Hz. This is desirable in cluttered environments where re-plotting the global path may lead to a more optimal trajectory to the goal.

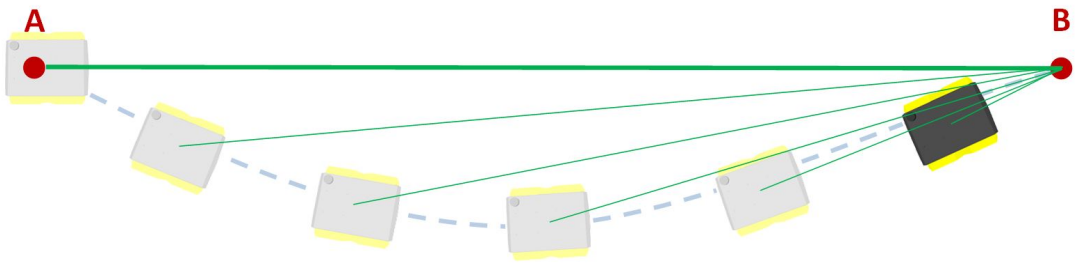


FIGURE B.4: Illustration of how path re-plotting can result in curved trajectory

However, as illustrated in figure B.4, it also means that if there is drift in the motor controllers (as there was with the UGV used in this project) then the the robot will not follow the original global path (bold green) when travelling from point A to point B. Instead, it will move at an angle to the path, and as it moves the global path will be re-plotted *starting from the robot's current location* (thin green). This will continue until the robot is close enough to the goal that the local path overrides any drift in the system. This behaviour is typically not an issue indoors in obstacle-rich environments where the robot's trajectory will rarely be straight for long.

To overcome this, the `planner_frequency` parameter was set to 0.0 Hz. At this frequency the global planner will only run when a new goal is received or the local planner reports that its path is blocked. This means that the global path is not re-plotted and the local planner will not deviate from it significantly.

B.2.2 Autonomous Waypoint Navigation

As described in the previous section, the ROS *move_base* node can autonomously move the UGV from one point to another. However, this is not useful unless multiple move goals can be issued in sequence. To achieve this the `actionlib` package was used with a custom ROS node written specifically to manage it. This section describes how the node operates.

The `actionlib` package provides implementation for an *ActionClient*, which can issue *Goal* messages to an *ActionServer*. The ActionServer receives the goals, executes them, provides *Feedback* messages on request, and a *Result* message when the goal has been completed. The package includes several generic goal, feedback and result messages, one

of which is a move goal. The `move_base` node also provides an implementation of a simple ActionServer, so goal messages can be sent to it directly from an ActionClient.

For this project, a custom ROS node called `waypoint_manager` was written to act as an ActionClient that could issue move goals to the `move_base` node and receive feedback. A short summary of how the `waypoint_manager` node operates is provided below:

1. Read in a list of move goals from a .txt file, where each goal is a desired coordinate in the `map` frame (X_{goal} and Y_{goal})
2. Issue the first (or next) goal to the `move_base` node for the robot to pursue in the real-world.
3. Wait for the `move_base` node to return a result or wait for a watch-dog timer to expire. The length of the timer is determined by the UGV's speed and distance to the goal.
4. Repeat steps 3-4 until the end of the list of coordinates has been reached.

The inclusion of a timer ensures that if a particular move goal takes too long to get to, or is too hard to achieve, the UGV will simply move on to the next goal. In software, this process is similar to the pseudo code shown below:

```
goalList = read_input_coordinates(filename)
index = 0

while index < len(goalList):
    goalN = goalList[index]
    timeToGoal = calculate_eta(goalN)
    send_goal_to_move_base(goalN)
    index += 1

    # Cycle while waiting for result
    while (currentTime < timeToGoal) and (atGoal == False):
        atGoal = is_robot_at_goal(goalN)
```

The `waypoint_manager` node runs on a laptop connected to the UGV over an ad-hoc WiFi network, providing a means of remote control. The `waypoint_manager` node completes the action client/server pair with the `move_base` node, and is the last node needed for autonomous navigation. The final software structure is shown in figure B.5, which includes package names in braces where relevant. The exact structure and data flow is only approximate for illustration purposes.

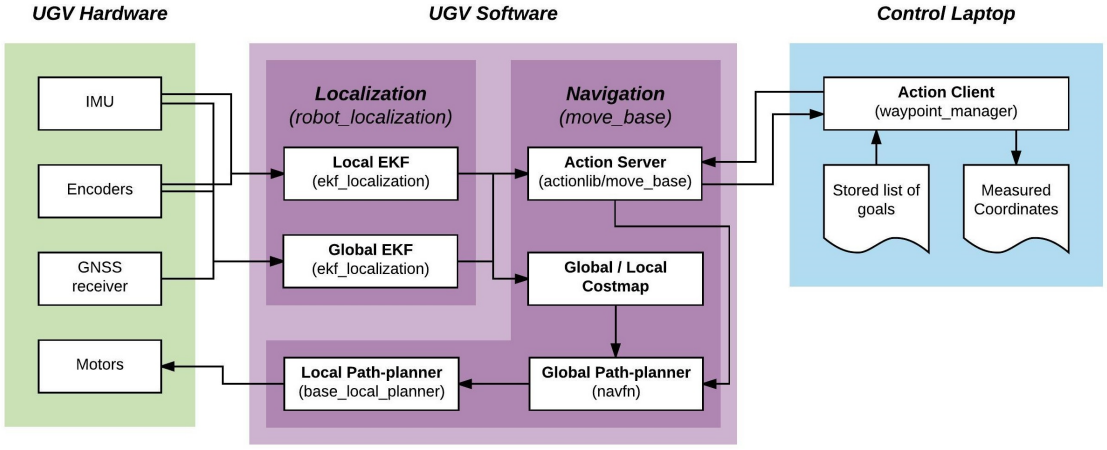


FIGURE B.5: Illustration of final software structure

This system allows the user to specify a complex path for the UGV to follow through any given environment. In the surveying context, this lets the user specify exactly how they want the UGV to travel across the survey area. Referring to Figure B.6 as an example, by issuing coordinates (denoted by red dots) in sequence a specific trajectory (shown as the black line) can be a achieved.

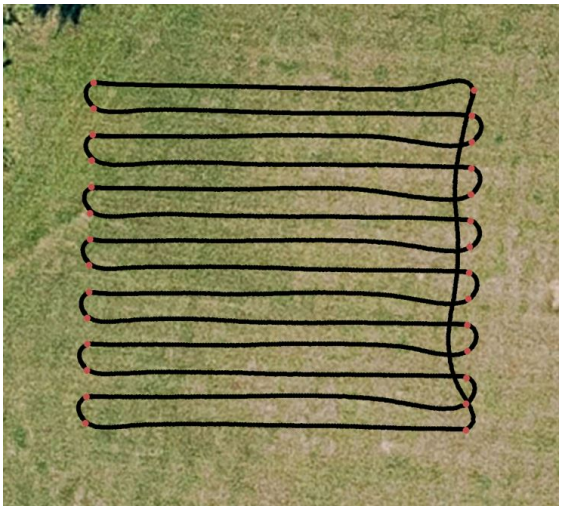


FIGURE B.6: Example trajectory from autonomous navigation across an outdoor area

This functionality gives the UGV real autonomy, as it allows the UGV to travel across an area between specified points without any human intervention. However, at this point the coordinates must still be specified in terms of the `map` frame, which is determined by the UGV's current location when the `ekf_localization` nodes are launched.

B.3 Development of Surveying Capability

The final stage of development was to devise a way for the UGV to record the survey coordinates of the terrain it travels over, to complete its required functionality and allow it to autonomously conduct a topographic survey. The following section describes how this functionality was achieved.

As explained in section 8.5(f), the UGV tracks it's location in the `map` reference frame, and a conversion is needed to change WGS84 or UTM coordinates in and out of the frame. For transforming move goals *into* the `map` frame, a simple rotation of the X, Y coordinates is enough. But for transforming survey coordinates *out* of the frame, a method is needed that preserves the vertical height or elevation. But before this conversion takes place, the UGV must have some "survey point" for reference.

B.3.1 Establishing a UGV survey point

To survey terrain the UGV must have some reference point that is, or is assumed to be, in contact with the ground. E.g. the tip of a survey rod or the end of a LiDAR laser beam. The only parts of the UGV chassis that contact the ground are the wheels, but because the wheels are not equipped with drop-sensors there is no way of knowing if the wheel is in direct contact with the ground. To solve this issue, an additional ROS reference frame called `survey_point` was created just below the `base_link` frame. This reference frame is situated between the four wheels at exactly the point where they are assumed to be in contact with the ground, as shown in figure B.7

It is assumed that if one wheel is off the ground, the change in elevation below that wheel is a local minima, and therefore the origin of the `survey_point` frame is still in approximate contact with the ground. So when the UGV records survey data, it is effectively just recording the position of the `survey_point` reference frame over time.

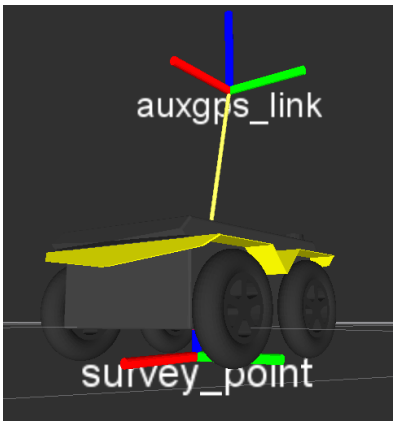


FIGURE B.7: Location of `survey_point` reference frame

The reference frame was added to the UGV's URDF file, so that it is automatically generated when the UGV is booted.